



Post Office Box 579, Pacific Grove, California 93950 (408) 648-3898

AN INTRODUCTION TO CP/M FEATURES AND FACILITIES

COPYRIGHT (c) 1976, 1977, 1978  
DIGITAL RESEARCH

REVISION OF JANUARY 1978

## Table of Contents

| Section                                    | Page |
|--------------------------------------------|------|
| 1. INTRODUCTION .....                      | 1    |
| 2. FUNCTIONAL DESCRIPTION OF CP/M .....    | 3    |
| 2.1. General Command Structure .....       | 3    |
| 2.2. File References .....                 | 3    |
| 3. SWITCHING DISKS .....                   | 6    |
| 4. THE FORM OF BUILT-IN COMMANDS .....     | 7    |
| 4.1. ERA afn cr .....                      | 7    |
| 4.2. DIR afn cr .....                      | 8    |
| 4.3. REN ufn1=ufn2 cr .....                | 8    |
| 4.4. SAVE n ufn cr .....                   | 9    |
| 4.5. TYPE ufn cr .....                     | 9    |
| 5. LINE EDITING AND OUTPUT CONTROL.....    | 11   |
| 6. TRANSIENT COMMANDS .....                | 12   |
| 6.1. STAT cr .....                         | 13   |
| 6.2. ASM ufn cr .....                      | 16   |
| 6.3. LOAD ufn cr .....                     | 17   |
| 6.4. PIP cr .....                          | 18   |
| 6.5. ED ufn cr .....                       | 25   |
| 6.6. SYSGEN cr .....                       | 27   |
| 6.7. SUBMIT ufn parm#1 ... parm#n cr ..... | 28   |
| 6.8. DUMP ufn cr .....                     | 30   |
| 6.9. MOVCPM cr .....                       | 30   |
| 7. BDOS ERROR MESSAGES .....               | 33   |
| 8. OPERATION OF CP/M ON THE MDS .....      | 34   |

## 1. INTRODUCTION.

CP/M is a monitor control program for microcomputer system development which uses IBM-compatible flexible disks for backup storage. Using a computer mainframe based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration which uses an Intel 8080 (or Zilog Z-80) Central Processing Unit, and has at least 64K bytes of main memory with up to four IBM-compatible diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in the Digital Research document entitled "CP/M System Alteration Guide." Although the standard Digital Research version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of distinct programs can be stored in both source and machine executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled with CP/M's Console Command Processor, the resulting facilities equal or excel similar large computer facilities.

CP/M is logically divided into several distinct parts:

|      |                                       |
|------|---------------------------------------|
| BIOS | Basic I/O System (hardware dependent) |
| BDOS | Basic Disk Operating System           |
| CCP  | Console Command Processor             |
| TPA  | Transient Program Area                |

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (teletype, CRT, Paper Tape Reader/Punch, and user-defined peripherals), and can be tailored by the user for any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies which provide fully dynamic file construction while minimizing head movement across the disk during access. Any particular file may contain any number of records, not exceeding the size of any single disk. In a standard CP/M system, each disk can contain up to 64 distinct files. The

BDOS has entry points which include the following primitive operations which can be programmatically accessed:

|        |                                                        |
|--------|--------------------------------------------------------|
| SEARCH | Look for a particular disk file by name.               |
| OPEN   | Open a file for further operations.                    |
| CLOSE  | Close a file after processing.                         |
| RENAME | Change the name of a particular file.                  |
| READ   | Read a record from a particular file.                  |
| WRITE  | Write a record onto the disk.                          |
| SELECT | Select a particular disk drive for further operations. |

The CCP provides symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands which are available in the CCP are listed in a following section.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs which are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

It should be mentioned that any or all of the CP/M component subsystems can be "overlayed" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlayed; thus, the user program need only branch to the bootstrap loader at the end of execution, and the complete CP/M monitor is reloaded from disk.

It should be reiterated that the CP/M operating system is partitioned into distinct modules, including the BIOS portion which defines the hardware environment in which CP/M is executing. Thus, the standard system can be easily modified to any non-standard environment by changing the peripheral drivers to handle the custom system.

## 2. FUNCTIONAL DESCRIPTION OF CP/M.

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks which are online (the standard system addresses up to four different disk drives). These disk drives are labelled A, B, C, and D. A disk is "logged in" if the CCP is currently addressing the disk. In order to clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start up, the CP/M system is brought in from disk A, and the CCP displays the message

xxK CP/M VER m.m

where xx is the memory size (in kilobytes) which this CP/M system manages, and m.m is the CP/M version number. All CP/M systems are initially set to operate in a 16K memory space, but can be easily reconfigured to fit any memory size on the host system (see the MOVCPM transient command). Following system signon, CP/M automatically logs in disk A, prompts the user with the symbol "A>" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

### 2.1. GENERAL COMMAND STRUCTURE.

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

|      |                                                 |
|------|-------------------------------------------------|
| ERA  | Erase specified files.                          |
| DIR  | List file names in the directory.               |
| REN  | Rename the specified file.                      |
| SAVE | Save memory contents in a file.                 |
| TYPE | Type the contents of a file on the logged disk. |

Nearly all of the commands reference a particular file or group of files. The form of a file reference is specified below.

### 2.2. FILE REFERENCES.

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references can be either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference may be

satisfied by a number of different files.

File references consist of two parts: the primary name and the secondary name. Although the secondary name is optional, it usually is generic; that is, the secondary name "ASM," for example, is used to denote that the file is an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a "." as shown below:

pppppppp.sss

where pppppppp represents the primary name of eight characters or less, and sss is the secondary name of no more than three characters. As mentioned above, the name

pppppppp

is also allowed and is equivalent to a secondary name consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

< > . , ; : = ? \* [ ]

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" may be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

X?Z.C?M

is satisfied by the unambiguous file names

XYZ.COM

and

XZ.CAM

Note that the ambiguous reference

\*.\*

is equivalent to the ambiguous file reference

?????????.???

while

and            pppppppp.\*  
              \*.sss

are abbreviations for

and            pppppppp.???  
              ??????????.sss

respectively. As an example,

DIR \*.\*

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

DIR X.Y

searches only for a file by the name X.Y Similarly, the command

DIR X?Y.C?M

causes a search for all (unambiguous) file names on the disk which satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

|     |         |         |
|-----|---------|---------|
| X   | XYZ     | GAMMA   |
| X.Y | XYZ.COM | GAMMA.1 |

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through Z followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

|           |         |         |
|-----------|---------|---------|
| A:X.Y     | B:XYZ   | C:GAMMA |
| Z:XYZ.COM | B:X.A?M | C:*.ASM |

It should also be noted that all alphabetic lower case letters in file and drive names are always translated to upper case when they are processed by the CCP.

### 3. SWITCHING DISKS.

The operator can switch the currently logged disk by typing the disk drive name (A, B, C, or D) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands shown below might occur after the CP/M system is loaded from disk A:

16K CP/M VER 1.4

A>DIR                      List all files on disk A.

SAMPLE    ASM

SAMPLE    PRN

A>B:                      Switch to disk B.

B>DIR \*.ASM              List all "ASM" files on B.

DUMP      ASM

FILES     ASM

B>A:                      Switch back to A.

#### 4. THE FORM OF BUILT-IN COMMANDS.

The file and device reference forms described above can now be used to fully specify the structure of the built-in commands. In the description below, assume the following abbreviations:

|     |   |                            |
|-----|---|----------------------------|
| ufn | - | unambiguous file reference |
| afn | - | ambiguous file reference   |
| cr  | - | carriage return            |

Further, recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabetic characters are treated as if they are upper case in command names and file references.

##### 4.1 ERA afn cr

The ERA (erase) command removes files from the currently logged-in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files which are erased are those which satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

|             |                                                                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ERA X.Y     | The file named X.Y on the currently logged disk is removed from the disk directory, and the space is returned.                                                                    |
| ERA X.*     | All files with primary name X are removed from the current disk.                                                                                                                  |
| ERA *.ASM   | All files with secondary name ASM are removed from the current disk.                                                                                                              |
| ERA X?Y.C?M | All files on the current disk which satisfy the ambiguous reference X?Y.C?M are deleted.                                                                                          |
| ERA *.*     | Erase all files on the current disk (in this case the CCP prompts the console with the message "ALL FILES (Y/N)?" which requires a Y response before files are actually removed). |
| ERA B:*.PRN | All files on drive B which satisfy the ambiguous reference ???????.PRN are deleted, independently of the currently logged disk.                                                   |

#### 4.2. DIR afn cr

The DIR (directory) command causes the names of all files which satisfy the ambiguous file name afn to be listed at the console device. As a special case, the command

DIR

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR \*.\*"). Valid DIR commands are shown below.

DIR X.Y

DIR X?Z.C?M

DIR ??,Y

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

DIR B:

DIR B:X.Y

DIR B:\*.A?M

If no files can be found on the selected diskette which satisfy the directory request, then the message "NOT FOUND" is typed at the console.

#### 4.3. REN ufn1=ufn2 cr

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn1). The CCP also allows the user to type a left-directed arrow instead of the equal sign, if the user's console supports this graphic character. Examples of the REN command are

REN X.Y=Q.R                      The file Q.R is changed to X.Y.

REN XYZ.COM=XYZ.XXX            The file XYZ.XXX is changed to XYZ.COM.

The operator can precede either ufn1 or ufn2 (or both) by an optional drive address. Given that ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive as ufn1. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to reside on that drive as well. If both ufn1 and ufn2 are preceded by drive names, then the same drive must be

TYPE X.PLM

TYPE XXX

The TYPE command expands tabs (ctrl-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name as shown below.

TYPE B:X.PRN

The file X.PRN from drive B is displayed.

## 5. LINE EDITING AND OUTPUT CONTROL.

The CCP allows certain line editing functions while typing command lines.

|        |                                                                                                              |
|--------|--------------------------------------------------------------------------------------------------------------|
| rubout | Delete and echo the last character typed at the console.                                                     |
| ctl-U  | Delete the entire line typed at the console.                                                                 |
| ctl-X  | (Same as ctl-U)                                                                                              |
| ctl-R  | Retype current command line: types a "clean line" following character deletion with rubouts.                 |
| ctl-E  | Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed. |
| ctl-C  | CP/M system reboot (warm start)                                                                              |
| ctl-Z  | End input from the console (used in PIP and ED).                                                             |

The control functions ctl-P and ctl-S affect console output as shown below.

|       |                                                                                                                                                                                                                                                                                      |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ctl-P | Copy all subsequent console output to the currently assigned list device (see the STAT command). Output is sent to both the list device and the console device until the next ctl-P is typed.                                                                                        |
| ctl-S | Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature is used to stop output on high speed consoles, such as CRT's, in order to view a segment of output before continuing. |

Note that the ctl-key sequences shown above are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines can generally be up to 255 characters in length; they are not acted upon until the carriage return key is typed.

## 6. TRANSIENT COMMANDS.

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands defined for execution under the CCP are shown below. Additional functions can easily be defined by the user (see the LOAD command definition).

|        |                                                                                                                                                                                            |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| STAT   | List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.                |
| ASM    | Load the CP/M assembler and assemble the specified program from disk.                                                                                                                      |
| LOAD   | Load the file in Intel "hex" machine code format and produce a file in machine executable form which can be loaded into the TPA (this loaded program becomes a new command under the CCP). |
| DDT    | Load the CP/M debugger into TPA and start execution.                                                                                                                                       |
| PIP    | Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.                                                                                       |
| ED     | Load and execute the CP/M text editor program.                                                                                                                                             |
| SYSGEN | Create a new CP/M system diskette.                                                                                                                                                         |
| SUBMIT | Submit a file of commands for batch processing.                                                                                                                                            |
| DUMP   | Dump the contents of a file in hex.                                                                                                                                                        |
| MOVCPM | Regenerate the CP/M system for a particular memory size.                                                                                                                                   |

Transient commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. As an added convenience, the transient command can be preceded by a drive name, which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

B:STAT

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

The basic transient commands are listed in detail below.

#### 6.1. STAT cr

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

```
STAT cr
STAT "command line" cr
```

Special forms of the "command line" allow the current device assignment to be examined and altered as well. The various command lines which can be specified are shown below, with an explanation of each form shown to the right.

STAT cr

If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

```
x: R/W, SPACE: nnnK
or
x: R/O, SPACE: nnnK
```

for each active drive x, where R/W indicates the drive may be read or written, and R/O indicates the drive is read only (a drive becomes R/O by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive x is given in kilobytes by nnn.

STAT x: cr

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

```
BYTES REMAINING ON B: nnnK
```

STAT afn cr

The command line can also specify a set of files to be scanned by STAT. The files which satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

```
RECS BYTS EX D:FILENAME.TYP
rrrr bbbK ee d:pppppppp.sss
```

where rrrr is the number of 128-byte records

allocated to the file, bbb is the number of kilo-  
bytes allocated to the file ( $bbb=rrrr*128/1024$ ),  
ee is the number of 16K extensions ( $ee=bbb/16$ ),  
d is the drive name containing the file (A...Z),  
pppppppp is the (up to) eight-character primary  
file name, and sss is the (up to) three-character  
secondary name. After listing the individual  
files, the storage usage is summarized.

STAT x:afn cr

As a convenience, the drive name can be given  
ahead of the afn. In this case, the specified  
drive is first selected, and the form "STAT afn"  
is executed.

STAT x:=R/O cr

This form sets the drive given by x to read-only,  
which remains in effect until the next warm or  
cold start takes place. When a disk is read-only,  
the message

BDOS ERR ON x: READ ONLY

will appear if there is an attempt to write to  
the read-only disk x. CP/M waits until a key  
is depressed before performing an automatic warm  
start (at which time the disk becomes R/W).

The STAT command also allows control over the physical to logical device  
assignment (see the IOBYTE function described in the manuals "CP/M Interface  
Guide" and "CP/M System Alteration Guide"). In general, there are four  
logical peripheral devices which are, at any particular instant, each assigned  
to one of several physical peripheral devices. The four logical devices are  
named:

|      |                                                                                |
|------|--------------------------------------------------------------------------------|
| CON: | The system console device (used by CCP<br>for communication with the operator) |
| RDR: | The paper tape reader device                                                   |
| PUN: | The paper tape punch device                                                    |
| LST: | The output list device                                                         |

The actual devices attached to any particular computer system are driven  
by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device,  
for example, could actually be a high speed reader, Teletype reader, or  
cassette tape. In order to allow some flexibility in device naming and  
assignment, several physical devices are defined, as shown below:

|      |                                                                                   |
|------|-----------------------------------------------------------------------------------|
| TTY: | Teletype device (slow speed console)                                              |
| CRT: | Cathode ray tube device (high speed console)                                      |
| BAT: | Batch processing (console is current RDR:,<br>output goes to current LST: device) |
| UC1: | User-defined console                                                              |
| PTR: | Paper tape reader (high speed reader)                                             |
| UR1: | User-defined reader #1                                                            |
| UR2: | User-defined reader #2                                                            |
| PTP: | Paper tape punch (high speed punch)                                               |
| UP1: | User-defined punch #1                                                             |
| UP2: | User-defined punch #2                                                             |
| LPT: | Line printer                                                                      |
| UL1: | User-defined list device #1                                                       |

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device may be implemented as a cassette write operation, if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The possible logical to physical device assignments can be displayed by typing

```
STAT VAL: cr
```

The STAT prints the possible values which can be taken on for each logical device:

```
CON. = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command

```
STAT DEV: cr
```

which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON: = CRT:
RDR: = UR1:
PUN: = PTP:
LST: = TTY:
```

The current logical to physical device assignment can be changed by typing a STAT command of the form

```
STAT ld1 = pd1, ld2 = pd2 , ... , ldn = pdn cr
```

where ld1 through ldn are logical device names, and pd1 through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

```
STAT CON:=CRT: cr
STAT PUN: = TTY:,LST:=LPT:, RDR:=TTY: cr
```

## 6.2. ASM ufn cr

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements where the secondary name is assumed to be ASM, and thus is not specified. The following ASM commands are valid:

```
ASM X
```

```
ASM GAMMA
```

The two-pass assembler is automatically executed. If assembly errors occur during the second pass, the errors are printed at the console.

The assembler produces a file

```
x.PRN
```

where x is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file can be listed

at the console using the TYPE command, or sent to a peripheral device using PIP (see the PIP command structure below). Note also that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). Thus, the PRN file can serve as a backup for the original source file: if the source file is accidentally removed or destroyed, the PRN file can be edited (see the ED operator's guide) by removing the leftmost 16 characters of each line (this can be done by issuing a single editor "macro" command). The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

x.HEX

is also produced which contains 8080 machine language in Intel "hex" format suitable for subsequent loading and execution (see the LOAD command). For complete details of CP/M's assembly language program, see the "CP/M Assembler Language (ASM) User's Guide."

Similar to other transient commands, the source file for assembly can be taken from an alternate disk by prefixing the assembly language file name by a disk drive name. Thus, the command

ASM B:ALPHA cr

loads the assembler from the currently logged drive and operates upon the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

### 6.3. LOAD ufn cr

The LOAD command reads the file ufn, which is assumed to contain "hex" format machine code, and produces a memory image file which can be subsequently executed. The file name ufn is assumed to be of the form

x.HEX

and thus only the name x need be specified in the command. The LOAD command creates a file named

x.COM

which marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the file name x immediately after the prompting character ">" printed by the CCP.

In general, the CCP reads the name x following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

x.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by simply typing the primary name. In this way, the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which can be deleted at the user's option.) The operation can take place on an alternate drive if the file name is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

It must be noted that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) which begin at 100H, the beginning of the TPA. Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA. Programs which occupy regions of memory other than the TPA can be loaded under DDT.

#### 6.4. PIP cr

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

- (1) PIP cr
- (2) PIP "command line" cr

In both cases, PIP is loaded into the TPA and executed. In case (1), PIP reads command lines directly from the console, prompted with the "\*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ... , source#n cr

where "destination" is the file or peripheral device to receive the data, and

"source#1, ..., source#n" represents a series of one or more files or devices which are copied from left to right to the destination.

When multiple files are given in the command line (i.e, n > 1), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the O parameter to override this assumption). The equal symbol (=) can be replaced by a left-oriented arrow, if your console supports this ASCII character, to improve readability. Lower case ASCII alphabets are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines which exceed the console width).

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A:, B:, C:, or D:) which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

|                              |                                                                                                                                 |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| X = Y cr                     | Copy to file X from file Y,<br>where X and Y are unambiguous<br>file names; Y remains unchanged.                                |
| X = Y,Z cr                   | Concatenate files Y and Z and<br>copy to file X, with Y and Z<br>unchanged.                                                     |
| X.ASM=Y.ASM,Z.ASM,FIN.ASM cr | Create the file X.ASM from the<br>concatenation of the Y, Z, and<br>FIN files with type ASM.                                    |
| NEW.ZOT = B:OLD.ZAP cr       | Move a copy of OLD.ZAP from drive<br>B to the currently logged disk;<br>name the file NEW.ZOT.                                  |
| B:A.U = B:B.V,A:C.W,D:X cr   | Concatenate file B.V from drive B<br>with C.W from drive A and D.X.<br>from the logged disk; create<br>the file A.U on drive B. |

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are

PIP x:=afn cr

PIP x:=y:afn cr

PIP ufn = y: cr

PIP x:ufn = y: cr

The first form copies all files from the currently logged disk which satisfy the afn to the same file names on drive x (x = A...Z). The second form is equivalent to the first, where the source for the copy is drive y (y = A...Z). The third form is equivalent to the command "PIP ufn=y:ufn cr" which copies the file given by ufn from drive y to the file ufn on drive x. The fourth form is equivalent to the third, where the source disk is explicitly given by y.

Note that the source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn which satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

|                   |                                                                                       |
|-------------------|---------------------------------------------------------------------------------------|
| B:=*.COM cr       | Copy all files which have the secondary name "COM" to drive B from the current drive. |
| A:=B:ZAP.* cr     | Copy all files which have the primary name "ZAP" to drive A from drive B.             |
| ZAP.ASM=B: cr     | Equivalent to ZAP.ASM=B:ZAP.ASM                                                       |
| B:ZOT.COM=A: cr   | Equivalent to B:ZOT.COM=A:ZOT.COM                                                     |
| B:=GAMMA.BAS cr   | Same as B:GAMMA.BAS=GAMMA.BAS                                                         |
| B:=A:GAMMA.BAS cr | Same as B:GAMMA.BAS=A:GAMMA.BAS                                                       |

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

CON: (console), RDR: (reader), PUN: (punch), and LST: (list)

while the physical devices are

TTY: (console, reader, punch, or list)  
 CRT: (console, or list), UCL: (console)  
 PTR: (reader), URL: (reader), UR2: (reader)  
 PTP: (punch), UPL: (punch), UP2: (punch)  
 LPT: (list), ULL: (list)

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. (The current physical device mapping is defined by IOBYTE; see the "CP/M Interface Guide" for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are

NUL: Send 40 "nulls" (ASCII 0's) to the device  
 (this can be issued at the end of punched output).

EOF: Send a CP/M end-of-file (ASCII ctl-Z) to the destination device (sent automatically at the end of all ASCII data transfers through PIP).

INP: Special PIP input source which can be "patched" into the PIP program itself: PIP gets the input data character-by-character by CALLing location 103H, with data returned in location 109H (parity bit must be zero).

OUT: Special PIP output destination which can be patched into the PIP program: PIP CALLs location 106H with data in register C for each character to transmit. Note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT (see the DDT operator's manual).

PRN: Same as LST:, except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines, with an initial eject (same as [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been

read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created (\$\$\$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a rubout suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the re-read, type a single carriage return at the console, and PIP will attempt another read. If the tape position cannot be properly read, simply continue the read (by typing a return following the error message), and enter the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is a RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard, then the read operation is terminated normally.

Valid PIP commands are shown below.

|                            |                                                                                                                                                     |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| PIP LST: = X.PRN cr        | Copy X.PRN to the LST device and terminate the PIP program.                                                                                         |
| PIP cr                     | Start PIP for a sequence of commands (PIP prompts with "***").                                                                                      |
| *CON:=X.ASM,Y.ASM,Z.ASM cr | Concatenate three ASM files and copy to the CON device.                                                                                             |
| *X.HEX=CON:,Y.HEX,PTR: cr  | Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y.HEX, followed by data from PTR until a ctl-Z is encountered. |
| *cr                        | Single carriage return stops PIP.                                                                                                                   |

PIP PUN:=NUL:,X.ASM,EOF:,NUL: cr      Send 40 nulls to the punch device;  
                                          then copy the X.ASM file to the  
                                          punch, followed by an end-of-file  
                                          (ctl-Z) and 40 more null charac-  
                                          ters.

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

- B      Block mode transfer: data is buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).
- Dn     Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.
- E      Echo all transfer operations to the console as they are being performed.
- F      Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.
- H      Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.
- I      Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).
- L      Translate upper case alphabetic to lower case.
- N      Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is

set.

- O Object file (non-ASCII) transfer: the normal CP/M end of file is ignored.
  - Pn Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.
  - Qs<sup>†</sup>z Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.
  - Ss<sup>†</sup>z Start copying from the source device when the string s is encountered (terminated by ctl-Z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.
- NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.
- (1) PIP cr
  - (2) PIP "command line" cr
- Tn Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.
  - U Translate lower case alphabets to upper case during the the copy operation.
  - V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).
  - Z Zero the parity bit on input for each ASCII character.

The following are valid PIP commands which specify parameters in the file transfer:

- PIP X.ASM=B:[v] cr      Copy X.ASM from drive B to the current drive and verify that the data was properly copied.
- PIP LPT:=X.ASM[nt8u] cr      Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case.

PIP PUN:=X.HEX[i],Y.ZOT[h] cr     First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

PIP X.LIB = Y.ASM [ sSUBR1:'z qJMP L3'z ] cr     Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBR1:" has been found, and quit copying after the string "JMP L3" is encountered.

PIP PRN:=X.ASM[p50]             Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p60 is the assumed parameter list for a PRN file; p50 overrides the default value.

#### 6.5. ED ufn cr

The ED program is the CP/M system context editor, which allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given in the ED user's manual, "ED: a Context Editor for the CP/M Disk System." In general, ED allows the operator to create and operate upon source files which are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory), which is instead defined by the number of characters typed between cr's. The ED program has a number of commands for character string searching, replacement, and insertion, which are useful in the creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 16K CP/M system), the file size which can be edited is not limited, since data is easily "paged" through this work area.

Upon initiation, ED creates the specified source file, if it does not exist, and opens the file for access. The programmer then "appends" data from the source file into the work area, if the source file already exists (see the A command), for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

ED X.ASM cr

- the ED program creates an intermediate work file with the name

X.\$\$\$

- to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version, and renaming the previous version. Suppose, for example, that the current X.ASM file was improperly edited; the sequence of CCP command shown below would reclaim the backup file.

|                 |                                          |
|-----------------|------------------------------------------|
| DIR X.*         | Check to see that BAK file is available. |
| ERA X.ASM       | Erase most recent version.               |
| REN X.ASM=X.BAK | Rename the BAK file to ASM.              |

Note that the operator can abort the edit at any point (reboot, power failure, ctrl-C, or Q command) without destroying the original file. In this case, the BAK file is not created, and the original file is always intact.

The ED program also allows the user to "ping-pong" the source and create backup files between two disks. The form of the ED command in this case is

ED ufn d:

where ufn is the name of a file to edit on the currently logged disk, and d is the name of an alternate drive. The ED program reads and processes the source file, and writes the new file to drive d, using the name ufn. Upon completion of processing, the original file becomes the backup file. Thus, if the operator is addressing disk A, the following command is valid:

ED X.ASM B:

which edits the file X.ASM on drive A, creating the new file X.\$\$\$ on drive B. Upon completion of a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.\$\$\$ is renamed to B:X.ASM. For user convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file by the name B:X.ASM exists before the editing begins, the message

FILE EXISTS

is printed at the console as a precaution against accidentally destroying a source file. In this case, the operator must first ERASE the existing file and then restart the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are shown below

|               |                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ED A:X.ASM    | Edit the file X.ASM on drive A, with new file and backup on drive A.                                                                                                            |
| ED B:X.ASM A: | Edit the file X.ASM on drive B to the temporary file X.\$\$\$ on drive A. On termination of editing, change X.ASM on drive B to X.BAK, and change X.\$\$\$ on drive A to X.ASM. |

#### 6.6. SYSGEN cr

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands, with interaction as shown below.

|                                              |                                                                                                                                                                                                                                                     |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SYSGEN cr                                    | Initiate the SYSGEN program.                                                                                                                                                                                                                        |
| SYSGEN VERSION m.m                           | SYSGEN sign-on message.                                                                                                                                                                                                                             |
| SOURCE DRIVE NAME (OR RETURN TO SKIP)        | Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system; usually A. If a copy of CP/M already exists in memory, due to a MOVCPM command, type a cr only. Typing a drive name x will cause the response: |
| SOURCE ON x THEN TYPE RETURN                 | Place a diskette containing the CP/M operating system on drive x (x is one of A, B, C, or D). Answer with cr when ready.                                                                                                                            |
| FUNCTION COMPLETE                            | System is copied to memory. SYSGEN will then prompt with:                                                                                                                                                                                           |
| DESTINATION DRIVE NAME (OR RETURN TO REBOOT) | If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name x will cause SYSGEN to prompt                                   |

with:

DESTINATION ON x THEN TYPE RETURN Place new diskette into drive  
x; type return when ready.

FUNCTION COMPLETE New diskette is initialized  
in drive x.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system, and only the built-in commands are available. A factory-fresh IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

PIP B: = A: \*.\*[v] or

which copies all files from disk drive A to disk drive B, and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

It should be noted that a SYSGEN does not destroy the files which already exist on a diskette; it results only in construction of a new operating system. Further, if a diskette is being used only on drives B through D, and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place. In fact, a new diskette needs absolutely no initialization to be used with CP/M.

#### 6.7. SUBMIT ufn parm#1 ... parm#n or

The SUBMIT command allows CP/M commands to be batched together for automatic processing. The ufn given in the SUBMIT command must be the filename of a file which exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands, with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed "\$" parameters of the form

```
$1 $2 $3 ... $n
```

corresponding to the number of actual parameters which will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters \$1 ... \$n in the prototype commands. If the number of formal and actual parameters does not correspond, then the submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

```
$$$SUB
```

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input, rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. Further, the user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case, the \$\$\$SUB file is removed, and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs which execute under CP/M can abort processing of command files when error conditions occur by simply erasing any existing \$\$\$SUB file.

In order to introduce dollar signs into a SUBMIT file, the user may type a "\$\$" which reduces to a single "\$" within the command file. Further, an up-arrow symbol "^" may precede an alphabetic character x, which produces a single ctl-x character within the file.

The last command in a SUB file can initiate another SUB file, thus allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

and the command

```
SUBMIT ASMBL X PRN cr
```

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of \$1 and "PRN" for all occurrences of \$2, resulting in a \$\$\$SUB file containing the commands

```

ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN

```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file which is on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon, however, when they appear on drive A. Thus, it is possible to create a submitted file on drive B which is executed at a later time when it is inserted in drive A.

#### 6.8. DUMP ufn cr

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in the "CP/M Interface Guide" as an example of a program written for the CP/M environment.)

#### 6.9. MOVCPM cr

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters may be used to indicate (1) the desired size of the new system and (2) the disposition of the new system at program termination. If the first parameter is omitted or a "\*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "\*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are:

MOVCPM cr

Relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). Upon completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system which is constructed contains a BIOS for the Intel MDS 800.

|               |                                                                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| MOVCPM n cr   | Create a relocated CP/M system for management of an n kilobyte system (n must be in the range 16 to 64), and execute the system, as described above.  |
| MOVCPM * * cr | Construct a relocated memory image for the current memory configuration, but leave the memory image in memory, in preparation for a SYSGEN operation. |
| MOVCPM n * cr | Construct a relocated memory image for an n kilobyte memory system, and leave the memory image in preparation for a SYSGEN operation.                 |

The command

MOVCPM \* \*

for example, constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

READY FOR "SYSGEN" OR  
"SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the current memory size in kilobytes. The operator can then type

|                                              |                                                                                                                                       |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| SYSGEN cr                                    | Start the system generation.                                                                                                          |
| SOURCE DRIVE NAME (OR RETURN TO SKIP)        | Respond with a cr to skip the CP/M read operation since the system is already in memory as a result of the previous MOVCPM operation. |
| DESTINATION DRIVE NAME (OR RETURN TO REBOOT) | Respond with B to write new system to the diskette in drive B. SYSGEN will prompt with:                                               |
| DESTINATION ON B, THEN TYPE RETURN           | Ready the fresh diskette on drive B and type a return when ready.                                                                     |

Note that if you respond with "A" rather than "B" above, the system will be written to drive A rather than B. SYSGEN will continue to type the prompt:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

until the operator responds with a single carriage return, which stops the

SYSGEN program with a system reboot.

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user could have typed

SAVE 32 CPMxx.COM

at the completion of the MOVCPM function, which would place the CP/M memory image on the currently logged disk in a form which can be "patched." This is necessary when operating in a non-standard environment where the BIOS must be altered for a particular peripheral device configuration, as described in the "CP/M System Alteration Guide."

Valid MOVCPM commands are given below:

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| MOVCPM 48 cr   | Construct a 48K version of CP/M and start execution.                                                                               |
| MOVCPM 48 * cr | Construct a 48K version of CP/M in preparation for permanent recording; response is<br>READY FOR "SYSGEN" OR<br>"SAVE 32CPM48.COM" |
| MOVCPM * * cr  | Construct a maximum memory version of CP/M and start execution.                                                                    |

It is important to note that the newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Digital Research Software Licensing Agreement.

## 7. BDOS ERROR MESSAGES.

There are three error situations which the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

BDOS ERR ON x: error

where x is the drive name, and "error" is one of the three error messages:

BAD SECTOR  
SELECT  
READ ONLY

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally due to a malfunctioning disk controller, or an extremely worn diskette. If you find that your system reports this error more than once a month, you should check the state of your controller electronics, and the condition of your media. You may also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturer's equipment will produce the "BAD SECTOR" message when read by the MDS. In any case, recovery from this condition is accomplished by typing a `ctl-C` to reboot (this is the safest!), or a return, which simply ignores the bad sector in the file operation. Note, however, that typing a return may destroy your diskette integrity if the operation is a directory write, so make sure you have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the A through D range. In this case, the value of x in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette which has been designated as read-only in a `STAT` command, or has been set to read-only by the BDOS. In general, the operator should reboot CP/M either by using the warm start procedure (`ctl-C`) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read-only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. Upon issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

## 8. OPERATION OF CP/M ON THE MDS.

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. A basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labelled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the device which is to be taken as the system console, and the light should go out (if it does not, then check connections and baud rates). The BOOT switch is then turned off, and the CP/M signon message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands.

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates a RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Note, however, that the user must not remove a diskette and replace it with another without rebooting the system (cold or warm start), unless the inserted diskette is "read only."

Due to hardware hang-ups or malfunctions, CP/M may type the message

BDOS ERR ON x: BAD SECTOR

where x is the drive which has a permanent error. This error may occur when drive doors are opened and closed randomly, followed by disk operations, or may be due to a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring re-initialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off, to avoid random transients which often make their way to the drive electronics.

It should be noted that factory-fresh IBM-compatible diskettes should be used rather than diskettes which have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces non-standard sector numbering throughout the diskette. This non-standard numbering seriously degrades the performance of CP/M, and will operate noticeably slower.

than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M which causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

---

Note: "MDS 800" and "ISIS" are registered trademarks of Intel Corporation.



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

**CP/M 2 USER'S GUIDE**

**COPYRIGHT (c) 1979  
DIGITAL RESEARCH**

## CP/M 2 USER'S GUIDE

Copyright (c) 1979  
Digital Research, Box 579  
Pacific Grove, California

|                                                        |    |
|--------------------------------------------------------|----|
| 1. An Overview of CP/M 2.0 Facilities . . . . .        | 1  |
| 2. User Interface . . . . .                            | 3  |
| 3. Console Command Processor (CCP) Interface . . . . . | 4  |
| 4. STAT Enhancements . . . . .                         | 5  |
| 5. PIP Enhancements . . . . .                          | 6  |
| 6. ED Enhancements . . . . .                           | 10 |
| 7. The XSUB Function . . . . .                         | 11 |
| 8. BDOS Interface Conventions . . . . .                | 12 |
| 9. CP/M 2.0 Memory Organization . . . . .              | 27 |
| 10. BIOS Differences . . . . .                         | 28 |

## 1. AN OVERVIEW OF CP/M 2.0 FACILITIES.

CP/M 2.0 is a high-performance single-console operating system which uses table driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All of the fundamental file restrictions are removed, while maintaining upward compatibility from previous versions of release 1. Features of CP/M 2.0 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability to expand to thirty-two megabytes in future releases. The directory size can be field configured to contain any reasonable number of entries, and each file is optionally tagged with read/only and system attributes. Users of CP/M 2.0 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2.0 which provide direct access to any of the 65536 records of an eight megabyte file.

All disk-dependent portions of CP/M 2.0 are placed into a BIOS-resident "disk parameter block" which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2.0. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2.0 operation. Deblocking information is also provided which aids in assembly or disassembly of sector sizes which are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general-purpose subroutines which use the this deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table driven data access algorithms, make CP/M 2.0 truly a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2.0 is structured, however, so that as much as 128K bytes of data is addressed by a single physical extent (corresponding to a single directory entry), thus maintaining compatibility with previous versions while taking full advantage of directory space.

Random access facilities are present in CP/M 2.0 which allow immediate reference to any record of an eight megabyte file. Using CP/M's unique data organization, data blocks are only allocated when actually required and movement to a record position requires little search time. Sequential file access is upward compatible from earlier versions to the full eight megabytes, while random access compatibility stops at 512K byte files. Due to CP/M 2.0's simpler and faster random access, application programmers are encouraged to alter their programs to take full advantage of the 2.0 facilities.

Several CP/M 2.0 modules and utilities have improvements which correspond to the enhanced file system. STAT and PIP both account for file attributes and user areas, while the CCP provides a "login"

(All Information Contained Herein is Proprietary to Digital Research.)

function to change from one user area to another. The CCP also formats directory displays in a more convenient manner and accounts for both CRT and hard-copy devices in its enhanced line editing functions.

The sections below point out the individual differences between CP/M 1.4 and CP/M 2.0, with the understanding that the reader is either familiar with CP/M 1.4, or has access to the 1.4 manuals. Additional information dealing with CP/M 2.0 I/O system alteration is presented in the Digital Research manual "CP/M 2.0 Alteration Guide."

(All Information Contained Herein is Proprietary to Digital Research.)

## 2. USER INTERFACE.

Console line processing takes CRT-type devices into account with three new control characters, shown with an asterisk in the list below (the symbol "ctl" below indicates that the control key is simultaneously depressed):

- rub/del removes and echoes last character
- ctl-C reboot when at beginning of line
- ctl-E physical end of line
- ctl-H backspace one character position\*
- ctl-J (line feed) terminates current input\*
- ctl-M (carriage return) terminates input
- ctl-R retype current line after new line
- ctl-U remove current line after new line
- ctl-X backspace to beginning of current line\*

In particular, note that ctl-H produces the proper backspace overwrite function (ctl-H can be changed internally to another character, such as delete, through a simple single byte change). Further, the line editor keeps track of the current prompt column position so that the operator can properly align data input following a ctl-U, ctl-R, or ctl-X command.

(All Information Contained Herein is Proprietary to Digital Research.)

### 3. CONSOLE COMMAND PROCESSOR (CCP) INTERFACE.

There are four functional differences between CP/M 1.4 and CP/M 2.0 at the console command processor (CCP) level. The CCP now displays directory information across the screen (four elements per line), the USER command is present to allow maintenance of separate files in the same directory, and the actions of the "ERA \*.\*" and "SAVE" commands have changed. The altered DIR format is self-explanatory, while the USER command takes the form:

USER n

where n is an integer value in the range 0 to 15. Upon cold start, the operator is automatically "logged" into user area number 0, which is compatible with standard CP/M 1.4 directories. The operator may issue the USER command at any time to move to another logical area within the same directory. Drives which are logged-in while addressing one user number are automatically active when the operator moves to another user number since a user number is simply a prefix which accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start operation when user 0 is again assumed.

Due to the fact that user numbers now tag individual directory entries, the ERA \*.\* command has a different effect. In version 1.4, this command can be used to erase a directory which has "garbage" information, perhaps resulting from use of a diskette under another operating system (heaven forbid!). In 2.0, however, the ERA \*.\* command affects only the current user number. Thus, it is necessary to write a simple utility to erase a nonsense disk (the program simply writes the hexadecimal pattern E5 throughout the disk).

The SAVE command in version 1.4 allows only a single memory save operation, with the potential of destroying the memory image due to directory operations following extent boundary changes. Version 2.0, however, does not perform directory operations in user data areas after disk writes, and thus the SAVE operation can be used any number of times without altering the memory image.

(All Information Contained Herein is Proprietary to Digital Research.)

#### 4. STAT ENHANCEMENTS.

The STAT program has a number of additional functions which allow disk parameter display, user number display, and file indicator manipulation. The command:

STAT VAL:

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status   : DSK: d:DSK:
User Status    :USR:
Iobyte Assign:
(list of possible assignments)
```

which gives an instant summary of the possible STAT commands. The command form:

STAT d:filename.typ \$S

where "d:" is an optional drive name, and "filename.typ" is an unambiguous or ambiguous file name, produces the output display format:

| Size  | Recs | Bytes | Ext | Acc             |
|-------|------|-------|-----|-----------------|
| 48    | 48   | 6k    | 1   | R/O A:ED.COM    |
| 55    | 55   | 12k   | 1   | R/O (A:PIP.COM) |
| 65536 | 128  | 2k    | 2   | R/W A:X.DAT     |

where the \$S parameter causes the "Size" field to be displayed (without the \$S, the Size field is skipped, but the remaining fields are displayed). The Size field lists the virtual file size in records, while the "Recs" field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The "Bytes" field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time, and thus the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent (each of these extents, however, may contain unallocated "holes" even though they are added into the record count). The "Ext" field counts the number of logical 16K extents allocated to the file. Unlike version 1.4, the Ext count does not necessarily correspond to the number of directory entries given to the file, since there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry, depending upon allocation size (in a special case, there are actually 256K bytes which can be directly addressed by a physical extent).

The "Acc" field gives the R/O or R/W access mode, which is changed using the commands shown below. Similarly, the parentheses

(All Information Contained Herein is Proprietary to Digital Research.)

shown around the PIP.COM file name indicate that it has the "system" indicator set, so that it will not be listed in DIR commands. The four command forms

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file (or set of files) in a read-only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent read/write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The "filename.typ" may be ambiguous or unambiguous, but in either case, the files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by "d:" is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file result in a terminal BDOS message

```
Bdos Err on d: File R/O
```

The BDOS then waits for a console input before performing a subsequent warm start (a "return" is sufficient to continue). The command form

```
STAT d:DSK:
```

lists the drive characteristics of the disk named by "d:" which is in the range A:, B:, ..., P:. The drive characteristics are listed in the format:

```
d: Drive Characteristics
65536: 128 Byte record Capacity
8192: Kilobyte Drive Capacity
128: 32 Byte Directory Entries
0: Checked Directory Entries
1024: Records/ Extent
128: Records/ Block
58: Sectors/ Track
2: Reserved Tracks
```

where "d:" is the selected drive, followed by the total record capacity (65536 is an 8 megabyte drive), followed by the total capacity listed in Kilobytes. The directory size is listed next, followed by the "checked" entries. The number of checked entries is usually identical to the directory size for removable media, since this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, since the media is not changed without at least a cold or warm start. The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or

(All Information Contained Herein is Proprietary to Digital Research.)

128K in the example above). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks. For logical drives which share the same physical disk, the number of reserved tracks may be quite large, since this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

STAT DSK:

produces a drive characteristics table for all currently active drives. The final STAT command form is

STATUSR:

which produces a list of the user numbers which have files on the currently addressed disk. The display format is:

```
Active User : 0
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In the above case, the active user number is 0 (default at cold start), with three user numbers which have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1, USER 2, or USER 3 commands, followed by a DIF command at the CCP level.

(All Information Contained Herein is Proprietary to Digital Research.)

## 5. PIP ENHANCEMENTS.

PIP provides three new functions which account for the features of CP/M 2.0. All three functions take the form of file parameters which are enclosed in square brackets following the appropriate file names. The commands are:

|    |                                                        |
|----|--------------------------------------------------------|
| Gn | Get File from User number n<br>(n in the range 0 - 15) |
| W  | Write over R/O files without<br>console interrogation  |
| R  | Read system files                                      |

The G command allows one user area to receive data files from another. Assuming the operator has issued the USER 4 command at the CCP level, the PIP statement

PIP X.Y = X.Y[G2]

reads file X.Y from user number 2 into user area number 4. The command

PIP A:=A:\*. \*[G2]

copies all of the files from the A drive directory for user number 2 into the A drive directory of the currently logged user number. Note that to ensure file security, one cannot copy files into a different area than the one which is currently addressed by the USER command.

Note also that the PIP program itself is initially copied to a user area (so that subsequent files can be copied) using the SAVE command. The sequence of operations shown below effectively moves PIP from one user area to the next.

|                   |                    |
|-------------------|--------------------|
| USER 0            | login user 0       |
| DDT PIP.COM       | load PIP to memory |
| (note PIP size s) |                    |
| G0                | return to CCP      |
| USER 3            | login user 3       |
| SAVE s PIP.COM    |                    |

where s is the integral number of memory "pages" (256 byte segments) occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the "NEXT" display. If for example, the next available address is 1D00, then PIP.COM requires 1C hexadecimal pages (or 1 times 16 + 12 = 28 pages), and thus the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can then be copied to another disk belonging to the same user number through normal pip transfers.

Under normal operation, PIP will not overwrite a file which is set to a permanent R/O status. If attempt is made to overwrite a R/O file, the prompt

(All Information Contained Herein is Proprietary to Digital Research.)

DESTINATION FILE IS R/O, DELETE (Y/N)?

is issued. If the operator responds with the character "y" then the file is overwritten. Otherwise, the response

**\*\* NOT DELETED \*\***

is issued, the file transfer is skipped, and PIP continues with the next operation in sequence. In order to avoid the prompt and response in the case of R/O file overwrite, the command line can include the *w* parameter, as shown below

PIP A:=B:\*.COM[w]

which copies all non-system files to the A drive from the B drive, and overwrites any R/O files in the process. If the operation involves several concatenated files, the *w* parameter need only be included with the last file in the list, as shown in the following example

PIP A.DAT = B.DAT,F:NEW.DAT,G:OLD.DAT[w]

Files with the system attribute can be included in PIP transfer if the *R* parameter is included, otherwise system files are not recognized. The command line

PIP ED.COM = B:ED.COM[R]

for example, reads the ED.COM file from the B drive, even if it has been marked as a R/O and system file. The system file attributes are copied, if present.

It should be noted that downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by use of 0. If compatibility is required with non-standard (e.g., "double density") versions of 1.4, it may be necessary to select 1. compatibility mode when constructing the internal disk parameter block (see the "CP/M 2.0 Alteration Guide," and refer to Section 10 which describes BIOS differences).

(All Information Contained Herein is Proprietary to Digital Research.)

## 6. ED ENHANCEMENTS.

The CP/M standard program editor provides several new facilities in the 2.0 release. Experience has shown that most operators use the relative line numbering feature of ED, and thus the editor has the "v" (Verify Line) option set as an initial value. The operator can, of course, disable line numbering by typing the "-v" command. If you are not familiar with the ED line number mode, you may wish to refer to the Appendix in the ED user's guide, where the "v" command is described.

ED also takes file attributes into account. If the operator attempts to edit a read/only file, the message

```
** FILE IS READ/ONLY **
```

appears at the console. The file can be loaded and examined, but cannot be altered in any way. Normally, the operator simply ends the edit session, and uses STAT to change the file attribute to R/W. If the edited file has the "system" attribute set, the message

```
"SYSTEM" FILE NOT ACCESSIBLE
```

is displayed at the console, and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

Finally, the insert mode ("i") command allows CRT line editing functions, as described in Section 2, above.

(All Information Contained Herein is Proprietary to Digital Research.)

## 7. THE XSUB FUNCTION.

An additional utility program is supplied with version 2.0 of CP/M, called XSUB, which extends the power of the SUBMIT facility to include line input to programs as well as the console command processor. The XSUB command is included as the first line of your submit file and, when executed, self-relocates directly below the CCP. All subsequent submit command lines are processed by XSUB, so that programs which read buffered console input (BDOS function 10) receive their input directly from the submit file. For example, the file SAVER.SUB could contain the submit lines:

```
XSUB
DDT
IS1.HEX
R
G0
SAVE 1 $2.COM
```

with a subsequent SUBMIT command:

```
SUBMIT SAVER X Y
```

which substitutes X for \$1 and Y for \$2 in the command stream. The XSUB program loads, followed by DDT which is sent the command lines "IX.HEX" "R" and "G0" thus returning to the CCP. The final command "SAVE 1 Y.COM" is processed by the CCP.

The XSUB program remains in memory, and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent submit command streams do not require the XSUB, unless an intervening cold start has occurred. Note that XSUB must be loaded after DESPOOL, if both are to run simultaneously.

(All Information Contained Herein is Proprietary to Digital Research.)

## 8. BDOS INTERFACE CONVENTIONS.

CP/M 2.0 system calls take place in exactly the same manner as earlier versions, with a call to location 0005H, function number in register C, and information address in register pair DE. Single byte values are returned in register A, with double byte values returned in HL (for reasons of compatibility, register A = L and register H = H upon return in all cases). A list of CP/M 2.0 calls is given below, with an asterisk following functions which are either new or revised from version 1.4 to 2.0. Note that a zero value is returned for out-of range function numbers.

|     |                       |     |                      |
|-----|-----------------------|-----|----------------------|
| 0   | System Reset          | 19* | Delete File          |
| 1   | Console Input         | 20  | Read Sequential      |
| 2   | Console Output        | 21  | Write Sequential     |
| 3   | Reader Input          | 22* | Make File            |
| 4   | Punch Output          | 23* | Rename File          |
| 5   | List Output           | 24* | Return Login Vector  |
| 6*  | Direct Console I/O    | 25  | Return Current Disk  |
| 7   | Get I/O Byte          | 26  | Set DMA Address      |
| 8   | Set I/O Byte          | 27  | Get Addr(Alloc)      |
| 9   | Print String          | 28* | Write Protect Disk   |
| 10* | Read Console Buffer   | 29* | Get Addr(R/O Vector) |
| 11  | Get Console Status    | 30* | Set File Attributes  |
| 12* | Return Version Number | 31* | Get Addr(Disk Parms) |
| 13  | Reset Disk System     | 32* | Set/Get User Code    |
| 14  | Select Disk           | 33* | Read Random          |
| 15* | Open File             | 34* | Write Random         |
| 16  | Close File            | 35* | Compute File Size    |
| 17* | Search for First      | 36* | Set Random Record    |
| 18* | Search for Next       |     |                      |

(Functions 28, 29, and 32 should be avoided in application programs to maintain upward compatibility with MP/M.) The new or revised functions are described below.

### Function 6: Direct Console I/O.

Direct Console I/O is supported under CP/M 2.0 for those applications where it is necessary to avoid the BDOS console I/O operations. Programs which currently perform direct I/O through the BIOS should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or register E contains an ASCII character. If the input value is FF, then function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, then function 6 assumes that E contains a valid ASCII character which is sent to the console.

(All Information Contained Herein is Proprietary to Digital Research.)

Function 10: Read Console Buffer.

The console buffer read operation remains unchanged except that console line editing is supported, as described in Section 2. Note also that certain functions which return the carriage to the leftmost position (e.g., `ctl-X`) do so only to the column position where the prompt ended (previously, the carriage returned to the extreme left margin). This new convention makes operator data input and line correction more legible.

Function 12: Return Version Number.

Function 12 has been redefined to provide information which allows version-independent programming (this was previously the "lift head" function which returned `HL=0000` in version 1.4, but performed no operation). The value returned by function 12 is a two-byte value, with `H = 00` for the CP/M release (`H = 01` for MP/M), and `L = 00` for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register `L`, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, you can write application programs which provide both sequential and random access functions, with random access disabled when operating under early releases of CP/M.

In the file operations described below, `DE` addresses a file control block (FCB). Further, all directory operations take place in a reserved area which does not affect write buffers as was the case in version 1.4, with the exception of Search First and Search Next, where compatibility is required.

The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, and a series of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at `005CH` can be used for random access files, since bytes `007DH`, `007EH`, and `007FH` are available for this purpose. For notational purposes, the FCB format is shown with the following fields:

(All Information Contained Herein is Proprietary to Digital Research.)

```
-----
|dr|fl|f2|/ /|f3|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|r1|r2|
-----
00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

where

dr            drive code (0 - 16)  
              0 => use default drive for file  
              1 => auto disk select drive A,  
              2 => auto disk select drive B,  
              ...  
              16=> auto disk select drive P.

f1...f8      contain the file name in ASCII  
              upper case, with high bit = 0

t1,t2,t3     contain the file type in ASCII  
              upper case, with high bit = 0  
              t1', t2', and t3' denote the  
              bit of these positions,  
              t1' = 1 => Read/Only file,  
              t2' = 1 => SYS file, no DIR list

ex           contains the current extent number,  
              normally set to 00 by the user, but  
              in range 0 - 31 during file I/O

s1           reserved for internal system use

s2           reserved for internal system use, set  
              to zero on call to OPEN, MAKE, SEARCH

rc           record count for extent "ex,"  
              takes on values from 0 - 128

d0...dn      filled-in by CP/M, reserved for  
              system use

cr           current record to read or write in  
              a sequential file operation, normally  
              set to zero by user

r0,r1,r2     optional random record number in the  
              range 0-65535, with overflow to r2,  
              r0,r1 constitute a 16-bit value with  
              low byte r0, and high byte r1

#### Function 15: Open File.

The Open File operation is identical to previous definitions, with the exception that byte s2 is automatically zeroed. Note that previous versions of CP/M defined this byte as zero, but made no

(All Information Contained Herein is Proprietary to Digital Research.)

checks to assure compliance. Thus, the byte is cleared to ensure upward compatibility with the latest version, where it is required.

#### Function 17: Search for First.

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise a value of A equal to 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is  $A * 32$  (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

#### Function 18: Search for Next.

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

#### Function 19: Delete File.

The Delete File function removes files which match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the reference file or files could not be found, otherwise a value in the range 0 to 3 is returned.

(All Information Contained Herein is Proprietary to Digital Research.)

Function 22: Make File.

The Make File operation is identical to previous versions of CP/M, except that byte s2 is zeroed upon entry to the BDOS.

Function 23: Rename File.

The Actions of the file rename functions are the same as previous releases except that the value 255 is returned if the rename function is unsuccessful (the file to rename could not be found), otherwise a value in the range 0 to 3 is returned.

Function 24: Return Login Vector.

The login vector value returned by CP/M 2.0 is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A, and the high order bit of H corresponds to the sixteenth drive, labelled P. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 28: Write Protect Current Disk.

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold or warm start operation produces the message

Bdos Err on d: R/O

Function 29: Get R/O Vector.

Function 29 returns a bit vector in register pair HL which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

Function 30: Set File Attributes.

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2' above) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a

(All Information Contained Herein is Proprietary to Digital Research.)

match, and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

#### Function 31: Get Disk Parameter Block Address.

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

#### Function 32: Set or Get User Code.

An application program can change or interrogate the currently active user number by calling function 32. If register E = FF hexadecimal, then the value of the current user number is returned in register A, where the value is in the range 0 to 31. If register E is not FF, then the current user number is changed to the value of E (modulo 32).

#### Function 33: Read Random.

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M release 2.0 does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, in version 2.0, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an

(All Information Contained Herein is Proprietary to Digital Research.)

error code, as listed below, or the value 30 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

- 01 reading unwritten data
- 02 (not returned in random mode)
- 03 cannot close current extent
- 04 seek to unwritten extent
- 05 (not returned in read mode)
- 06 seek past physical end of disk

Error code 01 and 04 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is non-zero under the current 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

#### Function 34: Write Random.

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Again, sequential read or write operations can commence following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent

(All Information Contained Herein is Proprietary to Digital Research.)

switch as it does in sequential mode under either CP/M 1.4 or CP/M 2.0.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created due to directory overflow.

#### Function 35: Compute File Size.

When computing the size of a file, the DE register pair addresses an PCB in random mode format (bytes r0, r1, and r2 are present). The PCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536 in version 2.0. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, only the last record of an eight megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

#### Function 36: Set Random Record.

The Set Random Record function causes the BDOS to automatically produce the random record position from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are

(All Information Contained Herein is Proprietary to Digital Research.)

involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

This section is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are the simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity (ok, so the program's not so brief), the only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow,

(All Information Contained Herein is Proprietary to Digital Research.)

which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

```

;*****
;*
;* sample random access program for cp/m 2.0
;*
;*****
0100          org      100h      ;base of tba
;
0000 =        reboot   equ      0000h      ;system reboot
0005 =        bdos     equ      0005h      ;bdos entry point
;
0001 =        coninp   equ      1          ;console input function
0002 =        conout   equ      2          ;console output function
0009 =        bstring  equ      9          ;print string until '$'
000a =        rstring  equ      10         ;read console buffer
000c =        version  equ      12         ;return version number
000f =        openf    equ      15         ;file open function
0010 =        closef   equ      16         ;close function
0016 =        makef    equ      22         ;make file function
0021 =        readr    equ      33         ;read random
0022 =        writr    equ      34         ;write random
;
005c =        fcb      equ      005ch      ;default file control block
007d =        ranrec   equ      fcb+33     ;random record position
007f =        ranovf   equ      fcb+35     ;high order (overflow) byte
0080 =        buff     equ      0080h      ;buffer address
;
000d =        cr       equ      0dh        ;carriage return
000a =        lf       equ      0ah        ;line feed
;
;*****
;*
;* load SP, set-up file for random access
;*
;*****
0100 31bc0     lxi      sp,stack
;
;          version 2.0?
0103 0e0c     mvi      c,version
0105 cd050     call     bdos
0108 fe20     cpi      20h      ;version 2.0 or better?
010a d2160     jnc      versok
;          bad version, message and go back
010d 111b0     lxi      d,badver
0110 cdda0     call     print
0113 c3000     jmp      reboot
;
versok:
;          correct version for random access

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0116 0e0f      mvi      c,openf ;open default tcb
0118 115c0     lxi      d,fc0
011b cd050     call     bdos
011e 3c        inr      a      ;err 255 becomes zero
011f c2370     jnz      ready

;
;      cannot open file, so create it
0122 0e16      mvi      c,makef
0124 115c0     lxi      d,fc0
0127 cd050     call     bdos
012a 3c        inr      a      ;err 255 becomes zero
012b c2370     jnz      ready

;
;      cannot create file, directory full
012e 113a0     lxi      d,nospace
0131 c0da0     call     print
0134 c3000     jmp      reboot ;back to ccp

;*****
;*
;* loop back to "ready" after each command
;*
;*****
;
ready:
;      file is ready for processing
;
0137 cde50     call     readcom ;read next command
013a 227d0     shld     ranrec ;store input record#
013d 217f0     lxi      h,ranovf
0140 3600      mvi      m,0      ;clear high byte if set
0142 fe51      cpi      'Q'      ;quit?
0144 c2560     jnz      notq

;
;      quit processing, close file
0147 0e10      mvi      c,closef
0149 115c0     lxi      d,fc0
014c cd050     call     bdos
014f 3c        inr      a      ;err 255 becomes 0
0150 cab90     jz       error ;error message, retry
0153 c3000     jmp      reboot ;back to ccp

;*****
;*
;* end of quit command, process write
;*
;*****
notq:
;      not the quit command, random write?
0156 fe57      cpi      'w'
0158 c2890     jnz      notw

;
;      this is a random write, fill buffer until cr
015b 114d0     lxi      d,datmsg
015e c0da0     call     print ;data prompt

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0161 0e7f      mvi      c,127    ;up to 127 characters
0163 21800     lxi      h,buff  ;destination
; rloop:      ;read next character to buff
0166 c5        push     b        ;save counter
0167 e5        push     h        ;next destination
0168 cdc20     call     getchr   ;character to a
016b e1        pop      h        ;restore counter
016c c1        uop      b        ;restore next to fill
016d fe0d     cpi      cr       ;end of line?
016f ca780     jz       erloop
; not end, store character
0172 77        mov      m,a
0173 23        inx      h        ;next to fill
0174 0d        ucr      c        ;counter goes down
; 0175 c2660     jnz      rloop   ;end of buffer?
erloop:
; end of read loop, store 00
0178 3600     mvi      m,0
;
; write the record to selected record number
017a 0e22     mvi      c,writer
017c 115c0     lxi      d,fcbl
017f cd050     call     bdos
0182 b7        ora      a        ;error code zero?
0183 c2b90     jnz      error    ;message if not
0186 c3370     jmp      ready    ;for another record
;
;*****
;*
;* end of write command, process read
;*
;*****
notw:
; not a write command, read record?
0189 fe52     cpi      'R'
018b c2b90     jnz      error    ;skip if not
;
; read random record
018e 0e21     mvi      c,readr
0190 115c0     lxi      d,fcbl
0193 cd050     call     bdos
0196 b7        ora      a        ;return code 00?
0197 c2b90     jnz      error
;
; read was successful, write to console
019a cdcf0     call     crlf     ;new line
019d 0e80     mvi      c,128    ;max 128 characters
019f 21800     lxi      h,buff  ;next to get
wloop:
01a2 7e        mov      a,m      ;next character
01a3 23        inx      h        ;next to get
01a4 e67f     ani      7fh      ;mask parity
01a6 ca370     jz       ready    ;for another command if 00
01a9 c5        push     b        ;save counter
01aa e5        push     h        ;save next to get

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

01ab fe20      cpi      ;graphic?
01ad d4c80     cnc      putchr ;skip output if not
01b0 e1        pop      h
01b1 c1        pop      b
01b2 0d        dcr      c      ;count=count-1
01b3 c2a20     jnz      wloop
01b6 c3370     jmp      ready

;
;*****
;
; * end of read command, all errors end-up here
;
;*****
;
error:
01b9 11590     lxi      d,errmsg
01bc cd0a0     call     print
01bf c3370     jmp      ready

;
;*****
;
; * utility subroutines for console i/o
;
;*****
;
getchr:
01c2 0e01     mvi      c,coninp
01c4 cd050     call     bdos
01c7 c9       ret

;
putchr:
01c8 0e02     mvi      c,conout
01ca 5f       mov      e,a      ;character to send
01cb cd050     call     bdos      ;send character
01ce c9       ret

;
crlf:
01cf 3e0d     mvi      a,cr      ;carriage return
01d1 cdc80     call     putchr
01d4 3e0a     mvi      a,lf      ;line feed
01d6 cdc80     call     putchr
01d9 c9       ret

;
print:
01da d5       push     d
01db cdcf0     call     crlf
01de d1       pop      d      ;new line
01df 0e09     mvi      c,pstring
01e1 cd050     call     bdos      ;print the string
01e4 c9       ret

;
readcom:

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

;read the next command line to the conbuf
01e5 1160d lxi d,prompt
01e8 cdda0 call print ;command?
01eb 0e0a mvi c,rstring
01ed 117a0 lxi d,conbuf
01f0 cd050 call bdos ;read command line
; command line is present, scan it
01f3 21000 lxi h,0 ;start with 0000
01f6 117c0 lxi d,conlin;command line
01f9 1a readc: ldax d ;next command character
01fa 13 inx d ;to next command position
01fb b7 ora a ;cannot be end of command
01fc c3 rz
; not zero, numeric?
01fd d630 sui '0'
01ff fe0a cpi 10 ;carry if numeric
0201 d2130 jnc endrd
; add-in next digit
0204 29 dad n ;*2
0205 40 mov c,1
0206 44 mov p,n ;bc = value * 2
0207 29 dad n ;*4
0208 29 dad n ;*8
0209 09 dad p ;*2 + *8 = *10
020a 85 add l ;+digit
020b 0c mov l,a
020c d2f90 jnc readc ;for another char
020f 24 inr n ;overflow
0210 c3f90 jmp readc ;for another char
) endrd:
; end of read, restore value in a
0213 c630 adi '0' ;command
0215 feb1 cpi 'a' ;translate case?
0217 d8 rc
; lower case, mask lower case bits
0218 e65f ani 101$1111b
021a c9 ret
;
;*****
;*
;* string data area for console messages
;*
;*****
badver:
021b 536f79 db 'sorry, you need cp/m version 2$'
nospace:
023a 4e6f29 db 'no directory space$'
datmsg:
024d 547970 db 'type data: $'
errmsg:
0259 457272 db 'error, try again.$'
prompt:
026b 4e6570 db 'next command? $'
;

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

;*****
;*
;* fixed and variable data area
;*
;*****
027a 21 conbuf: db      conlen ;length of console buffer
027b      consiz: ds      1      ;resulting size after read
027c      conlin: ds      32      ;length 32 buffer
0021 =    conlen equ      $-consiz
;
029c      ds      32      ;16 level stack
stack:
02bc      end

```

(All Information Contained Herein is Proprietary to Digital Research.)

## 9. CP/M 2.0 MEMORY ORGANIZATION.

Similar to earlier versions, CP/M 2.0 is field-altered to fit various memory sizes, depending upon the host computer memory configuration. Typical base addresses for popular memory sizes are shown in the table below.

| Module     | 20k    | 24k    | 32k    | 48k    | 64k   |
|------------|--------|--------|--------|--------|-------|
| CCP        | 3400H  | 4400H  | 6400H  | A400H  | E400H |
| BDOS       | 3C00H  | 4C00H  | 6C00H  | AC00H  | EC00H |
| BIOS       | 4A00H  | 5A00H  | 7A00H  | BA00H  | FA00H |
| Top of Ram | 4FFFFH | 5FFFFH | 7FFFFH | BFFFFH | FFFFH |

The distribution disk contains a CP/M 2.0 system configured for a 20k Intel MDS-800 with standard IBM 8" floppy disk drives. The disk layout is shown below:

| Sector | Track 00           | Module      | Track 01 | Module      |
|--------|--------------------|-------------|----------|-------------|
| 1      | (Bootstrap Loader) |             | 4080H    | BDOS + 480H |
| 2      | 3400H              | CCP + 000H  | 4100H    | BDOS + 500H |
| 3      | 3480H              | CCP + 080H  | 4180H    | BDOS + 580H |
| 4      | 3500H              | CCP + 100H  | 4200H    | BDOS + 600H |
| 5      | 3580H              | CCP + 180H  | 4280H    | BDOS + 680H |
| 6      | 3600H              | CCP + 200H  | 4300H    | BDOS + 700H |
| 7      | 3680H              | CCP + 280H  | 4380H    | BDOS + 780H |
| 8      | 3700H              | CCP + 300H  | 4400H    | BDOS + 800H |
| 9      | 3780H              | CCP + 380H  | 4480H    | BDOS + 880H |
| 10     | 3800H              | CCP + 400H  | 4500H    | BDOS + 900H |
| 11     | 3880H              | CCP + 480H  | 4580H    | BDOS + 980H |
| 12     | 3900H              | CCP + 500H  | 4600H    | BDOS + A00H |
| 13     | 3980H              | CCP + 580H  | 4680H    | BDOS + A80H |
| 14     | 3A00H              | CCP + 600H  | 4700H    | BDOS + B00H |
| 15     | 3A80H              | CCP + 680H  | 4780H    | BDOS + B80H |
| 16     | 3B00H              | CCP + 700H  | 4800H    | BDOS + C00H |
| 17     | 3B80H              | CCP + 780H  | 4880H    | BDOS + C80H |
| 18     | 3C00H              | BDOS + 000H | 4900H    | BDOS + D00H |
| 19     | 3C80H              | BDOS + 080H | 4980H    | BDOS + D80H |
| 20     | 3D00H              | BDOS + 100H | 4A00H    | BIOS + 000H |
| 21     | 3D80H              | BDOS + 180H | 4A80H    | BIOS + 080H |
| 22     | 3E00H              | BDOS + 200H | 4B00H    | BIOS + 100H |
| 23     | 3E80H              | BDOS + 280H | 4B80H    | BIOS + 180H |
| 24     | 3F00H              | BDOS + 300H | 4C00H    | BIOS + 200H |
| 25     | 3F80H              | BDOS + 380H | 4C80H    | BIOS + 280H |
| 26     | 4000H              | BDOS + 400H | 4D00H    | BIOS + 300H |

In particular, note that the CCP is at the same position on the disk, and occupies the same space as version 1.4. The BDOS portion, however, occupies one more 256-byte page and the BIOS portion extends through the remainder of track 01. Thus, the CCP is 800H (2048 decimal) bytes in length, the BDOS is E00H (3584 decimal) bytes in length, and the BIOS is up to 380H (898 decimal) bytes in length. In version 2.0, the BIOS portion contains the standard subroutines of 1.4, along with some initialized table space, as described in the following section.

(All Information Contained Herein is Proprietary to Digital Research.)

## 10. BIOS DIFFERENCES.

The CP/M 2.0 Basic I/O System differs only slightly in concept from its predecessors. Two new jump vector entry points are defined, a new sector translation subroutine is included, and a disk characteristics table must be defined. The skeletal form of these changes are found in the program shown below.

```

1:      org      4000h
2:      maclio   diskdef
3:      jmp      boot
4: ;
5:      jmp      listst ;list status
6:      jmp      sectran ;sector translate
7:      disks    4
8: ;
9:      bpb      equ      16*1024 ;bytes per block
10:     rpb      equ      bpb/128 ;records per block
11:     maxb     equ      65535/rpb ;max block number
12:     diskdef  0,1,58,3,bpb,maxb+1,128,0,2
13:     diskdef  1,1,58,,bpb,maxb+1,128,0,2
14:     diskdef  2,0
15:     diskdef  3,1
16: ;
17: boot:  ret      ;nop
18: ;
19: listst: xra     a          ;nop
20:      ret
21: ;
22: seldsk:
23:      ;drive number in c
24:      lxi      h,0          ;0000 in hl produces select error
25:      mov      a,c          ;a is disk number 0 ... ndisks-1
26:      cpi      ndisks       ;less than ndisks?
27:      rnc      ;return with HL = 0000 if not
28: ;      proper disk number, return dpb element address
29:      mov      l,c
30:      dad      h             ;*2
31:      dad      h             ;*4
32:      dad      h             ;*8
33:      dad      h             ;*16
34:      lxi      d,dpbase
35:      dad      d             ;HL=.dpb
36:      ret
37: ;
38: selsec:
39:      ;sector number in c
40:      lxi      h,sector
41:      mov      m,c
42:      ret
43: ;
44: sectran:
45:      ;translate sector BC using table at DE
46:      xchg      ;HL = .tran
47:      dad      b             ;single precision tran

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

48: ;      dad b again if double precision tran
49:      mov     l,m      ;only low byte necessary here
50: ;      fill both H and L if double precision tran
51:      ret              ;HL = ??ss
52: ;
53: sector: ds      1
54:      endef
55:      end

```

Referring to the program shown above, lines 3-6 represent the BIOS entry vector of 17 elements (version 1.4 defines only 15 jump vector elements). The last two elements provide access to the "LISTSR" (List Status) entry point for DESPOOL. The use of this particular entry point is defined in the DESPOOL documentation, and is no different than the previous 1.4 release. It should be noted that the 1.4 DESPOOL program will not operate under version 2.0, but an update version will be available from Digital Research in the near future.

The "SECTTRAN" (Sector Number Translate) entry shown in the jump vector at line 6 provides access to a BIOS-resident sector translation subroutine. This mechanism allows the user to specify the sector skew factor and translation for a particular disk system, and is described below.

A macro library is shown in the listing, called DISKDEF, included on line 2, and referenced in l2-l5. Although it is not necessary to use the macro library, it greatly simplifies the disk definition process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks. (See the CP/M 2.0 Alteration Guide for formulas which you can use to hand-code the tables produced by the DISKDEF library).

A BIOS disk definition consists of the following sequence of macro statements:

```

MACLIB  DISKDEF
.....
DISKS   n
DISKDEF 0,...
DISKDEF 1,...
.....
DISKDEF n-1
.....
ENDEF

```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate in-line

(All Information Contained Herein is Proprietary to Digital Research.)

fixed data tables, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

|     |                                              |
|-----|----------------------------------------------|
| dn  | is the logical disk number, 0 to n-1         |
| fsc | is the first physical sector number (0 or 1) |
| lsc | is the last sector number                    |
| skf | is the optional sector skew factor           |
| bls | is the data allocation block size            |
| dir | is the number of directory entries           |
| cks | is the number of "checked" directory entries |
| ofs | is the track offset to logical track 00      |
| [0] | is an optional 1.4 compatibility flag        |

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read-only so that data is not subsequently destroyed). Normally the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system

(All Information Contained Herein is Proprietary to Digital Research.)

space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only .6K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS      4
DISKDEF    0,1,26,6,1024,243,64,64,2
DISKDEF    1,0
DISKDEF    2,0
DISKDEF    3,0
```

```
.....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The definitions given in the program shown above (lines 12 through 15) provide access to the largest disks addressable by CP/M 2.0. All disks have identical parameters, except that drives 0 and 2 skip three sectors on every data access, while disks 1 and 3 access each sector in sequence as the disk revolves (there may, however, be a transparent hardware skew factor on these drives).

The DISKS macro generates n "disk header blocks," starting at address DPBASE which is a label generated by the macro. Each disk header block contains sixteen bytes, and correspond, in sequence, to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE EQU $
DPE0: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPE (disk parameter entry) labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the CP/M 2.0 Alteration Guide, but basically address the translation vector for the drive (all reference XLT0, which is the translation vector for drive 0 in the above example).

(All Information Contained Herein is Proprietary to Digital Research.)

followed by three 16-bit "scratch" addresses, followed by the directory buffer address, disk parameter block address, check vector address, and allocation vector address. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The SELDSK function is extended somewhat in version 2.0. In particular, the selected disk number is passed to the BIOS in register C, as before, and the SELDSK subroutine performs the appropriate software or hardware actions to select the disk. Version 2.0, however, also requires the SELDSK subroutine to return the address of the selected disk parameter header (DPE0, DPE1, DPE2, or DPE3, in the above example) in register HL. If SELDSK returns the value HL = 0000H, then the BDOS assumes the disk does not exist, and prints a select error message at the terminal. Program lines 22 through 36 give a sample CP/M 2.0 SELDSK subroutine, showing only the disk parameter header address calculation.

The subroutine SECTRAN is also included in version 2.0 which performs the actual logical to physical sector translation. In earlier versions of CP/M, the sector translation process was a part of the BDOS, and set to skip six sectors between each read. Due to differing rotational speeds of various disks, the translation function has become a part of the BIOS in version 2.0. Thus, the BDOS sends sequential sector numbers to SECTRAN, starting at sector number 0. The SECTRAN subroutine uses the sequential sector number to produce a translated sector number which is returned to the BDOS. The BDOS subsequently sends the translated sector number to SELSEC before the actual read or write is performed. Note that many controllers have the capability to record the sector skew on the disk itself, and thus there is no translation necessary. In this case, the "skf" parameter is omitted in the macro call, and SECTRAN simply returns the same value which it receives. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```

XLT0:  DB    1,7,13,19,25,5,11,17,23,3,9,15,21
        DB    2,8,14,20,26,6,12,18,24,4,10,16,22

```

If SECTRAN is required to translate a sector, then the following process takes place. The sector to translate is received in register pair BC. Only the C register is significant if the sector value does not exceed 255 (B = 00 in this case). Register pair DE addresses the sector translate table for this drive, determined by a previous call on SELDSK, corresponding to the first element of a disk parameter header (XLT0 in the case shown above). The SECTRAN subroutine then fetches the translated sector number by adding the input sector number to the base of the translate table, to get the indexed translate table address (see lines 46, 47, and 48 in the above program). The value at this location is then returned in register L. Note that if the number of sectors exceeds 255, the translate table contains 16-bit elements whose value must be returned in HL.

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS

(All Information Contained Herein is Proprietary to Digital Research.)

which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =      BEGDAT EQU $  
          (data areas)  
4DB0 =      ENDDAT EQU $  
013C =      DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

CP/M 2.0 is also easily adapted to disk subsystems whose sector size is a multiple of 128 bytes. Information is provided by the BCOS sector write operations which eliminates the need for pre-read operations, thus allowing blocking and deblocking to take place at the BIOS level.

See the "CP/M 2.0 Alteration Guide" for additional details concerning tailoring your CP/M system to your particular hardware.



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

**ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM  
USER'S MANUAL**

**COPYRIGHT (c) 1976, 1978  
DIGITAL RESEARCH**

## Table of Contents

|                                              |    |
|----------------------------------------------|----|
| 1. ED TUTORIAL . . . . .                     | 1  |
| 1.1 Introduction to ED . . . . .             | 1  |
| 1.2 ED Operation . . . . .                   | 1  |
| 1.3 Text Transfer Functions . . . . .        | 1  |
| 1.4 Memory Buffer Organization . . . . .     | 5  |
| 1.5 Memory Buffer Operation . . . . .        | 5  |
| 1.6 Command Strings . . . . .                | 7  |
| 1.7 Text Search and Alteration . . . . .     | 8  |
| 1.8 Source Libraries . . . . .               | 11 |
| 1.9 Repetitive Command Execution . . . . .   | 12 |
| 2. ED ERROR CONDITIONS . . . . .             | 13 |
| 3. CONTROL CHARACTERS AND COMMANDS . . . . . | 14 |

## ED USER'S MANUAL

### 1. ED TUTORIAL

#### 1.1. Introduction to ED.

ED is the context editor for CP/M, and is used to create and alter CP/M source files. ED is initiated in CP/M by typing

$$\text{ED} \left\{ \begin{array}{l} \langle \text{filename} \rangle \\ \langle \text{filename} \rangle . \langle \text{filetype} \rangle \end{array} \right\}$$

In general, ED reads segments of the source file given by  $\langle \text{filename} \rangle$  or  $\langle \text{filename} \rangle . \langle \text{filetype} \rangle$  into central memory, where the file is manipulated by the operator, and subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 1.

#### 1.2. ED Operation

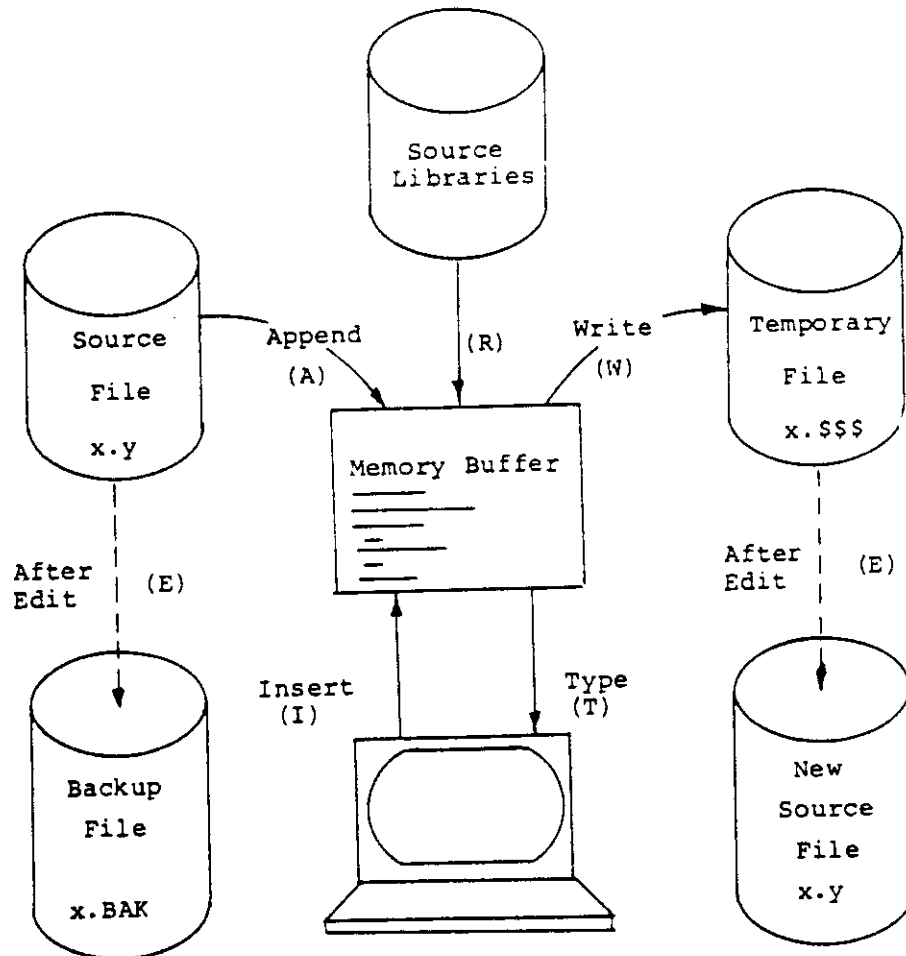
ED operates upon the source file, denoted in Figure 1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines which can be maintained in the memory buffer varies with the line length, but has a total capacity of about 6000 characters in a 16K CP/M system). Text material which has been edited is written onto a temporary work file under command of the operator. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from x.y to x.BAK so that the most recent previously edited source file can be reclaimed if necessary (see the CP/M commands ERASE and RENAME). The temporary file is then changed from x.\$\$\$ to x.y which becomes the resulting edited file.

The memory buffer is logically between the source file and working file as shown in Figure 2.

#### 1.3. Text Transfer Functions

Given that n is an integer value in the range 0 through 65535, the following ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file:

Figure 1. Overall ED Operation



Note: the ED program accepts both lower and upper case ASCII characters as input from the console. Single letter commands can be typed in either case. The U command can be issued to cause ED to translate lower case alphabets to upper case as characters are filled to the memory buffer from the console. Characters are echoed as typed without translation, however. The -U command causes ED to revert to "no translation" mode. ED starts with an assumed -U in effect.

Figure 2. Memory Buffer Organization

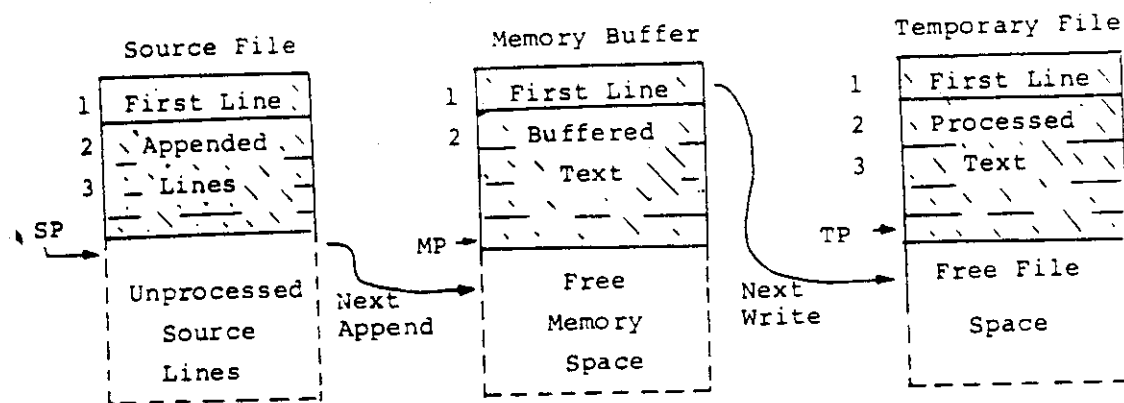
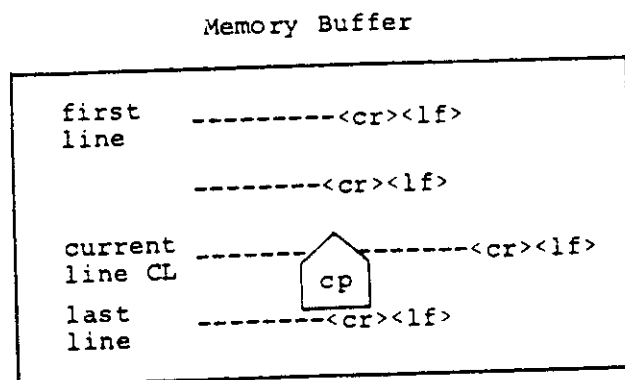


Figure 3. Logical Organization of Memory Buffer



- nA<cr>\* - append the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n.
  
- nW<cr> - write the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n+1 through MP to the top of the memory buffer. Increment TP by n.
  
- E<cr> - end the edit. Copy all buffered text to temporary file, and copy all unprocessed source lines to the temporary file. Rename files as described previously.
  
- H<cr> - move to head of new file by performing automatic E command. Temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created (equivalent to issuing an E command, followed by a reinvocation of ED using x.y as the file to edit).
  
- O<cr> - return to original file. The memory buffer is emptied, the temporary file id deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.
  
- Q<cr> - quit edit with no file alterations, return to CP/M.

There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write 1 line, respectively. In addition, if a pound sign (#) is given in the place of n, then the integer 65535 is assumed (the largest value for n which is allowed). Since most reasonably sized source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file. Two special forms of the A and W

---

\*<cr> represents the carriage-return key

commands are provided as a convenience. The command OA fills the current memory buffer to at least half-full, while OW writes lines until the buffer is at least half empty. It should also be noted that an error is issued if the memory buffer size is exceeded. The operator may then enter any command (such as W) which does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

#### 1.4. Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP which moves throughout the memory buffer under command of the operator. The memory buffer appears logically as shown in Figure 3 where the dashes represent characters of the source line of indefinite length, terminated by carriage-return (<cr>) and line-feed (<lf>) characters, and CP represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line which contains the CP.

#### 1.5. Memory Buffer Operation

Upon initiation of ED, the memory buffer is empty (ie, CP is both ahead and behind the first and last character). The operator may either append lines (A command) from the source file, or enter the lines directly from the console with the insert command

I<cr>

ED then accepts any number of input lines, where each line terminates with a <cr> (the <lf> is supplied automatically), until a control-z (denoted by +z is typed by the operator). The CP is positioned after the last character entered. The sequence

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
+z
```

leaves the memory buffer as shown below

`tnT<cr>` - If `n=0` then type the contents of the current line up to CP. If `n=1` then type the contents of the current line from CP to the end of the line. If `n>1` then type the current line along with `n-1` lines which follow, if `+` is specified. Similarly, if `n>1` and `-` is given, type the previous `n` lines, up to the CP. The break key can be depressed to abort long type-outs.

`tn<cr>` - equivalent to `tnLT`, which moves up or down and types a single line

#### 1.6. Command Strings

Any number of commands can be typed contiguously (up to the capacity of the CP/M console buffer), and are executed only after the `<cr>` is typed. Thus, the operator may use the CP/M console command functions to manipulate the input command:

|           |                                                                            |
|-----------|----------------------------------------------------------------------------|
| Rubout    | remove the last character                                                  |
| Control-U | delete the entire line                                                     |
| Control-C | re-initialize the CP/M System                                              |
| Control-E | return carriage for long lines without transmitting buffer (max 128 chars) |

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results shown to the right

| <u>Command String</u>          | <u>Effect</u>                                                             | <u>Resulting Memory Buffer</u>                                                                                                             |
|--------------------------------|---------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 1. <code>B2T&lt;cr&gt;</code>  | move to beginning of buffer and type 2 lines:<br>"NOW IS THE<br>TIME FOR" | <div> <div>CP</div> <div>NOW IS THE&lt;cr&gt;&lt;lf&gt;<br/>TIME FOR&lt;cr&gt;&lt;lf&gt;<br/>ALL GOOD MEN&lt;cr&gt;&lt;lf&gt;</div> </div> |
| 2. <code>5C0T&lt;cr&gt;</code> | move CP 5 characters and type the beginning of the line<br>"NOW I"        | <div> <div>NOW I</div> <div>CP</div> <div>S THE&lt;cr&gt;&lt;lf&gt;</div> </div>                                                           |

- |                                             |                                                                                   |                                                                                     |
|---------------------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| 3. 2L-T<cr>                                 | move two lines down<br>and type previous<br>line<br>"TIME FOR"                    | NOW IS THE<cr><lf><br>TIME FOR<cr><lf><br>ALL GOOD MEN<cr><lf>                      |
|                                             |  |                                                                                     |
| 4. -L#K<cr>                                 | move up one line,<br>delete 65535 lines<br>which follow                           | NOW IS THE<cr><lf>                                                                  |
|                                             |                                                                                   |  |
| 5. I<cr><br>TIME TO<cr><br>INSERT<cr><br>+z | insert two lines<br>of text                                                       | NOW IS THE<cr><lf><br>TIME TO<cr><lf><br>INSERT<cr><lf>                             |
|                                             |                                                                                   |  |
| 6. -2L#T<cr>                                | move up two lines,<br>and type 65535<br>lines ahead of CP<br>"NOW IS THE"         | NOW IS THE<cr><lf><br>TIME TO<cr><lf><br>INSERT<cr><lf>                             |
|                                             |                                                                                   |  |
| 7. <cr>                                     | move down one line<br>and type one line<br>"INSERT"                               | NOW IS THE<cr><lf><br>TIME TO<cr><lf><br>INSERT<cr><lf>                             |
|                                             |                                                                                   |  |

### 1.7. Text Search and Alteration

ED also has a command which locates strings within the memory buffer. The command takes the form

$$nF c_1 c_2 \dots c_k \left\{ \begin{array}{l} \langle cr \rangle \\ +z \end{array} \right\}$$

where  $c_1$  through  $c_k$  represent the characters to match followed by either a <cr> or control -z\*. ED starts at the current position of CP and attempts to match all  $k$  characters. The match is attempted  $n$  times, and if successful, the CP is moved directly after the character  $c_k$ . If the  $n$  matches are not successful, the CP is not moved from its initial position. Search strings can include +l (control-l), which is replaced by the pair of symbols <cr><lf>.

\*The control-z is used if additional commands will be typed following the +z.

The following commands illustrate the use of the F command:

| <u>Command String</u> | <u>Effect</u>                                                                                   | <u>Resulting Memory Buffer</u>                                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. B#T<cr>            | move to beginning and type entire buffer                                                        | <div style="display: inline-block; vertical-align: middle; text-align: center;">⌈cp⌋</div> NOW IS THE<cr><lf><br>TIME FOR<cr><lf><br>ALL GOOD MEN<cr><lf>  |
| 2. FS T<cr>           | find the end of the string "S T"                                                                | NOW IS T <div style="display: inline-block; vertical-align: middle; text-align: center;">⌈cp⌋</div> HE<cr><lf>                                             |
| 3. FI+zOTT            | find the next "I" and type to the CP then type the remainder of the current line:<br>"TIME FOR" | NOW IS THE<cr><lf><br>TI <div style="display: inline-block; vertical-align: middle; text-align: center;">⌈cp⌋</div> ME FOR<cr><lf><br>ALL GOOD MEN<cr><lf> |

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is:

I c<sub>1</sub>c<sub>2</sub>... c<sub>n</sub>+z      or

I c<sub>1</sub>c<sub>2</sub>... c<sub>n</sub><cr>

where c<sub>1</sub> through c<sub>n</sub> are characters to insert. If the insertion string is terminated by a +z, the characters c<sub>1</sub> through c<sub>n</sub> are inserted directly following the CP, and the CP is moved directly after character c<sub>n</sub>. The action is the same if the command is followed by a <cr> except that a <cr><lf> is automatically inserted into the text following character c<sub>n</sub>. Consider the following command sequences as examples of the F and I commands:

| <u>Command String</u> | <u>Effect</u>                                  | <u>Resulting Memory Buffer</u>                                                                                                                                     |
|-----------------------|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BITHIS IS +z<cr>      | Insert "THIS IS " at the beginning of the text | THIS IS NOW THE <cr><lf><br><div style="display: inline-block; vertical-align: middle; text-align: center;">⌈cp⌋</div><br>TIME FOR<cr><lf><br>ALL GOOD MEN<cr><lf> |

FTIME+z-4DIPLACE+z<cr>

find "TIME" and delete  
it; then insert "PLACE"

THIS IS NOW THE<cr><lf>  
PLACE  FOR<cr><lf>  
ALL GOOD MEN<cr><lf>

3FO+z-3D5DICHANGES+<cr>

find third occurrence  
of "O" (ie the second  
"O" in GOOD), delete  
previous 3 characters;  
then insert "CHANGES"

THIS IS NOW THE <cr><lf>  
PLACE FOR<cr><lf>  
ALL CHANGES  <cr><lf>

-8CISOURCE<cr> move back 8 characters  
and insert the line  
"SOURCE<cr><lf>"

THIS IS NOW THE<cr><lf>  
PLACE FOR<cr><lf>  
ALL SOURCE<cr><lf>  
 CHANGES<cr><lf>

ED also provides a single command which combines the F and  
I commands to perform simple string substitutions. The command  
takes the form

$$n \text{ S } c_1 c_2 \dots c_k + z d_1 d_2 \dots d_m \left\{ \begin{array}{c} \langle \text{cr} \rangle \\ + z \end{array} \right\}$$

and has exactly the same effect as applying the command string

$$F \text{ } c_1 c_2 \dots c_k + z - k D i d_1 d_2 \dots d_m \left\{ \begin{array}{c} \langle \text{cr} \rangle \\ + z \end{array} \right\}$$

a total of n times. That is, ED searches the memory buffer  
starting at the current position of CP and successively sub-  
stitutes the second string for the first string until the  
end of buffer, or until the substitution has been performed  
n times.

As a convenience, a command similar to F is provided by  
ED which automatically appends and writes lines as the search  
proceeds. The form is

$$n \text{ N } c_1 c_2 \dots c_k \left\{ \begin{array}{c} \text{cr} \\ + z \end{array} \right\}$$

which searches the entire source file for the nth occurrence  
of the string  $c_1 c_2 \dots c_k$  (recall that F fails if the string  
cannot be found in the current buffer). The operation of the

J command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory contents is written (ie, an automatic #W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command takes the form

$$n J c_1 c_2 \dots c_k \uparrow z d_1 d_2 \dots d_m \uparrow z e_1 e_2 \dots e_q \left\{ \begin{array}{l} \langle cr \rangle \\ \uparrow z \end{array} \right\}$$

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string  $c_1 c_2 \dots c_k$ . If found, insert the string  $d_1 d_2 \dots d_m$ , and move CP to follow  $d_m$ . Then delete all characters following CP up to (but not including) the string  $e_1 e_2 \dots e_q$ , leaving CP directly after  $d_m$ . If  $e_1 e_2 \dots e_q$  cannot be found, then no deletion is made. If the current line is

cp NOW IS THE TIME  $\langle cr \rangle \langle lf \rangle$

Then the command

JW  $\uparrow z$ WHAT  $\uparrow z \uparrow l \langle cr \rangle$

Results in

NOW WHAT cp  $\langle cr \rangle \langle lf \rangle$

(Recall that  $\uparrow l$  represents the pair  $\langle cr \rangle \langle lf \rangle$  in search and substitute strings).

It should be noted that the number of characters allowed by ED in the F, S, N, and J commands is limited to 100 symbols.

### 1.8. Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

$R f_1 f_2 \dots f_n \uparrow z$  or

$R f_1 f_2 \dots f_n \langle cr \rangle$

where  $f_1 f_2 \dots f_n$  is the name of a source file on the disk with as assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

$RMACRO \langle cr \rangle$

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file, and automatically inserts the characters into the memory buffer.

### 1.9. Repetitive Command Execution

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form:

$n M c_1 c_2 \dots c_k \left\{ \begin{array}{l} \langle cr \rangle \\ \uparrow z \end{array} \right\}$

where  $c_1 c_2 \dots c_k$  represent a string of ED commands, not including another M command. ED executes the command string  $n$  times if  $n > 1$ . If  $n = 0$  or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line which is changed:

$MFGAMMA \uparrow z - 5DIDELTA \uparrow z OTT \langle cr \rangle$

or equivalently

$MSGAMMA \uparrow z DELTA \uparrow z OTT \langle cr \rangle$

## 2. ED ERROR CONDITIONS

On error conditions, ED prints the last character read before the error, along with an error indicator:

|   |                                                                                                              |
|---|--------------------------------------------------------------------------------------------------------------|
| ? | unrecognized command                                                                                         |
| > | memory buffer full (use one of the commands D,K,N,S, or W to remove characters), F,N, or S strings too long. |
| # | cannot apply command the number of times specified (e.g., in F command)                                      |
| O | cannot open LIB file in R command                                                                            |

Cyclic redundancy check (CRC) information is written with each output record under CP/M in order to detect errors on subsequent read operations. If a CRC error is detected, CP/M will type

PERM ERR DISK d

where d is the currently selected drive (A,B,...). The operator can choose to ignore the error by typing any character at the console (in this case, the memory buffer data should be examined to see if it was incorrectly read), or the user can reset the system and reclaim the backup file, if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information:

TYPE x.BAK<cr>

where x is the file being edited. Then remove the primary file:

ERA x.y<cr>

and rename the BAK file:

REN x.y=x.BAK<cr>

The file can then be re-edited, starting with the previous version.

### 3. CONTROL CHARACTERS AND COMMANDS

The following table summarizes the control characters and commands available in ED:

| <u>Control Character</u> | <u>Function</u>                                     |
|--------------------------|-----------------------------------------------------|
| +c                       | system reboot                                       |
| +e                       | physical <cr><lf> (not actually entered in command) |
| +i                       | logical tab (cols 1,8,15,...)                       |
| +l                       | logical <cr><lf> in search and substitute strings   |
| +u                       | line delete                                         |
| +z                       | string terminator                                   |
| rubout                   | character delete                                    |
| break                    | discontinue command (e.g., stop typing)             |

| <u>Command</u> | <u>Function</u>                                             |
|----------------|-------------------------------------------------------------|
| nA             | append lines                                                |
| ±B             | begin bottom of buffer                                      |
| ±nC            | move character positions                                    |
| ±nD            | delete characters                                           |
| E              | end edit and close files<br>(normal end)                    |
| nF             | find string                                                 |
| H              | end edit, close and reopen<br>files                         |
| I              | insert characters                                           |
| nJ             | place strings in juxtaposition                              |
| ±nK            | kill lines                                                  |
| ±nL            | move down/up lines                                          |
| nM             | macro definition                                            |
| nN             | find next occurrence with<br>autoscan                       |
| O              | return to original file                                     |
| ±nP            | move and print pages                                        |
| Q              | quit with no file changes                                   |
| R              | read library file                                           |
| nS             | substitute strings                                          |
| ±nT            | type lines                                                  |
| ± U            | translate lower to upper case if U,<br>no translation if -U |
| nW             | write lines                                                 |
| nZ             | sleep                                                       |
| ±n<cr>         | move and type (±nLT)                                        |

## Appendix A: ED 1.4 Enhancements

The ED context editor contains a number of commands which enhance its usefulness in text editing. The improvements are found in the addition of line numbers, free space interrogation, and improved error reporting.

The context editor issued with CP/M 1.4 produces absolute line number prefixes when the "V" (Verify Line Numbers) command is issued. Following the V command, the line number is displayed ahead of each line in the format:

nnnnn:

where nnnnn is an absolute line number in the range 1 to 65535. If the memory buffer is empty, or if the current line is at the end of the memory buffer, then nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. Thus, the command

345:T

is interpreted as "move to absolute line 345, and type the line." Note that absolute line numbers are produced only during the editing process, and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute line number by a colon. Thus, the command

:400T

is interpreted as "type from the current line number through the line whose absolute number is 400." Combining the two line reference forms, the command

345::400T

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "0V", prints the memory buffer statistics in the form:

free/total

where "free" is the number of free bytes in the memory buffer (in decimal), and "total" is the size of the memory buffer.

ED 1.4 also includes a "block move" facility implemented through the "X" (Xfer) command. The form

nX

transfers the next n lines from the current line to a temporary file called

X\$\$\$\$\$\$\$.LIB

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred line accumulate one after another in this file, and can be retrieved by simply typing:

R

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred line file. That is, given that a set of lines has been transferred with the X command, they can be re-read any number of times back into the source file. The command

OX

is provided, however, to empty the transferred line file.

Note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted through cti-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

Due to common typographical errors, ED 1.4 requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

E (end), H (head), O (original), Q (quit)

must be typed as single letter commands.

ED 1.4 also prints error messages in the form

BREAK "x" AT c

where x is the error character, and c is the command where the error occurred.



CPM-86 PRELIMINARY DOCUMENTATION

COPYRIGHT (c) 1981

DIGITAL RESEARCH

CP/M<sup>TM</sup>-86 SYSTEM REFERENCE GUIDE

Copyright (c) 1980,1981

Digital Research  
P.O. Box 579  
801 Lighthouse Avenue  
Pacific Grove, CA 93950  
(408) 649-3896  
TWX 910 360 5001

All Rights Reserved

#### COPYRIGHT

Copyright (c) 1980 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

#### DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

#### TRADEMARKS

CP/M is a registered trademark of Digital Research. CP/NET, MP/M, LINK-80, RMAC and PL/I-80 are trademarks of Digital Research.

The "CP/M-86 SYSTEM REFERENCE GUIDE" was prepared using the Digital Research TEX Text Formatter.

\*\*\*\*\*  
\* First Printing: January 1981 \*  
\*\*\*\*\*

## TABLE OF CONTENTS

|      |                                                       |    |
|------|-------------------------------------------------------|----|
| 1.   | CP/M-86 SYSTEM OVERVIEW . . . . .                     | 1  |
| 1.1. | CP/M-86 General Characteristics . . . . .             | 1  |
| 1.2. | Specific CP/M-80 and CP/M-86 Differences . . . . .    | 3  |
| 2.   | COMMAND SETUP AND EXECUTION UNDER CP/M-86 . . . . .   | 5  |
| 2.1. | CCP Built-in and Transient Commands . . . . .         | 5  |
| 2.2. | Transient Program Execution Models . . . . .          | 6  |
| 2.3. | The 8080 Memory Model . . . . .                       | 7  |
| 2.4. | The Small Memory Model . . . . .                      | 8  |
| 2.5. | The Compact Memory Model . . . . .                    | 9  |
| 2.6. | Base Page Initialization . . . . .                    | 10 |
| 2.7. | Transient Program Setup and Termination . . . . .     | 11 |
| 3.   | COMMAND (CMD) FILE GENERATION . . . . .               | 12 |
| 3.1. | The Intel Hex File Format . . . . .                   | 12 |
| 3.2. | Operation of GENCMD . . . . .                         | 13 |
| 3.3. | Operation of LMCMD . . . . .                          | 15 |
| 3.4. | Command (CMD) File Format . . . . .                   | 15 |
| 4.   | BASIC DISK OPERATING SYSTEM FUNCTIONS . . . . .       | 19 |
| 4.1. | BDOS Parameters and Function Codes . . . . .          | 19 |
| 4.2. | Simple BDOS Calls . . . . .                           | 20 |
| 4.3. | BDOS File Operations . . . . .                        | 26 |
| 4.4. | BDOS Memory Management and Load Operations . . . . .  | 48 |
| 5.   | BASIC I/O SYSTEM (BIOS) ORGANIZATION . . . . .        | 55 |
| 5.1. | Organization of the BIOS . . . . .                    | 55 |
| 5.2. | The BIOS Jump Vector . . . . .                        | 56 |
| 5.3. | Simple Peripheral Devices . . . . .                   | 57 |
| 5.4. | BIOS Subroutine Entry Points . . . . .                | 59 |
| 6.   | BIOS DISK DEFINITION TABLES . . . . .                 | 65 |
| 6.1. | The Format of the Disk Parameter Tables . . . . .     | 65 |
| 6.2. | Automatic Table Generation Using GENDEF . . . . .     | 69 |
| 6.3. | GENDEF Output . . . . .                               | 73 |
| 7.   | CP/M-86 BOOTSTRAP AND ADAPTATION PROCEDURES . . . . . | 77 |
| 7.1. | The Cold Start Load Operation . . . . .               | 77 |
| 7.2. | Organization of CPM . . . . .                         | 80 |
| 8.   | SECTOR BLOCKING AND DEBLOCKING . . . . .              | 83 |
| 9.   | A SAMPLE RANDOM ACCESS PROGRAM . . . . .              | 85 |

## APPENDICES

|                                                          |     |
|----------------------------------------------------------|-----|
| Appendix A: Random Access Sample Program . . . . .       | 87  |
| Appendix B: Listing of the BOOT ROM . . . . .            | 93  |
| Appendix C: LDBIOS Listing . . . . .                     | 103 |
| Appendix D: BIOS Listing . . . . .                       | 109 |
| Appendix E: CBIOS Listing . . . . .                      | 125 |
| Appendix F: Blocking and Deblocking Algorithms . . . . . | 132 |

## 1. CP/M-86 SYSTEM OVERVIEW

This document presents the system programming aspects of CP/M-86, a single-user operating system for the Intel 8086 and compatible 8088 16-bit microprocessors. The discussion assumes basic familiarity with CP/M-80, the Digital Research 8-bit operating system, and presents an overview of CP/M-86 program interface conventions and adaptation procedures. CP/M-80 and CP/M-86 are equivalent at the user interface level and thus the Digital Research manuals

An Introduction to CP/M Features and Facilities  
ED: A Context Editor for the CP/M Disk System  
CP/M 2 User's Guide

along with the ASM86 and DDT86 User's Guides, which accompany this manual, should be referenced for further definitions and operating details. This document, in effect, replaces the CP/M 2 Interface Guide and CP/M 2 Alteration Guide for operation of CP/M with the 8086 and 8088 processors.

### 1.1. CP/M-86 General Characteristics.

CP/M-86 contains all facilities of CP/M-80 with additional features to account for increased processor address space of up to a megabyte (1,048,576) of main memory. Further, CP/M-86 maintains file compatibility with all previous versions of CP/M. The file structure of version 2 of CP/M is used, allowing as many as sixteen drives with up to eight megabytes on each drive. Thus, CP/M-80 and CP/M-86 files are completely interchangeable.

CP/M-86 resides in the file CPM.SYS which is loaded into memory by a cold start loader during system initialization. The cold start loader itself resides on the first two tracks of the system disk. CPM.SYS contains three program modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and user-configurable Basic I/O System (BIOS). The CCP and BDOS portions occupy approximately 10K bytes, while the size of the BIOS varies with the implementation. The operating system executes in any portion of memory above the reserved interrupt locations, while the remainder of the address space is partitioned into as many as eight non-contiguous regions, as defined in a BIOS table. Unlike CP/M-80, the CCP area cannot be used as a data area subsequent to transient program load: the CCP, BDOS, and BIOS remain in memory at all times, and are not reloaded upon each warm start.

Similar to CP/M-80, CP/M-86 loads and executes memory image files from disk. Memory image files are preceded by a "header record," defined in this document, which provides information required for proper program loading and execution. Memory image files under CP/M-86 are identified by a "CMD" file type.

All Information Presented Here is Proprietary to Digital Research

Unlike CP/M-80, CP/M-86 does not use absolute locations for system entry or default variables. The BDOS entry takes place through a software interrupt, while entry to the BIOS is provided by a new BDOS call. Two variables maintained in low memory under CP/M-80, the default disk number and I/O Byte, are placed in the CCP and BIOS, respectively. The final dependence upon absolute addresses is removed in CP/M-86 by maintaining initial "base page" values, such as the default FCB and default command buffer, in the transient program data area.

Utility programs such as ED, PIP, STAT and SUBMIT operate in the same manner under CP/M-86 and CP/M-80. In its operation, DDT86 resembles DDT supplied with CP/M-80 but instead allows interactive debugging of 8086 and 8088 machine code. Similarly, ASM86 allows assembly language programming and development for the 8086 and 8088 mnemonics using the Intel format. The GENCMD (Generate CMD) utility replaces the LOAD program of CP/M-80, and converts Intel 8086 "hex" files into memory image format suitable for execution under CP/M-86. Further, the LDCOPY (Loader Copy) program replaces SYSGEN, and is used to copy the cold start loader from a system disk for replication. In addition, a variation of GENCMD, called LMCMD, converts output from the Intel Locate utility into CMD format. Finally, GENDEF (Generate DISKDEF) is provided as an aid in producing custom disk parameter tables. ASM86, GENCMD, LMCMD, and GENDEF are also supplied in "COM" file format for cross-development under CP/M-80.

Several common abbreviations are used throughout this manual as defined below:

|                    |                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------|
| Nibble             | 4-bit half-byte                                                                                            |
| Byte               | 8-bit value                                                                                                |
| Word               | 16-bit value                                                                                               |
| Double Word        | 32-bit value                                                                                               |
| Paragraph          | 16 contiguous bytes                                                                                        |
| Paragraph Boundary | An address divisible evenly by 16 (low order nibble 0)                                                     |
| Segment            | Up to 64 contiguous bytes                                                                                  |
| Segment Register   | One of CS, DS, ES, or SS                                                                                   |
| Offset             | 16-bit displacement from a segment register                                                                |
| Group              | A segment-register-relative relocatable program unit                                                       |
| Address            | The effective memory address derived from the composition of a segment register value with an offset value |

A group consists of a set of segments which are loaded into memory as a single unit. Since a group may consist of more than 64K bytes, it is the responsibility of the application program to manage segment registers when code or data beyond the first 64K segment is accessed.

All Information Presented Here is Proprietary to Digital Research

## 1.2. Specific CP/M-80 and CP/M-86 Differences.

The structure of CP/M-86 is as close to CP/M-80 as possible in order to provide a familiar programming environment which allows application programs to be transported to the 8086 and 8088 processors with minimum effort. This section points out the specific differences between CP/M-80 and CP/M-86 in order to reduce your time in scanning this manual if you are already familiar with CP/M. The terms and concepts presented in this section are explained in detail throughout this manual, so you will need to refer to the Table of Contents to find relevant sections which provide specific definitions and information.

Due to the nature of the 8086, the fundamental difference between CP/M-80 and CP/M-86 is found in the management of the various relocatable groups. Although CP/M-80 references absolute memory locations by necessity, CP/M-86 takes advantage of the static relocation inherent in the 8086 processor. The operating system itself is usually loaded directly above the interrupt locations, at location 0400H, and relocatable transient programs load directly above the operating system. However, you can load CP/M-86 into any portion of memory without changing the operating system (thus, there is no MOVCPM utility with CP/M-86), and transient programs will load and run in any non-reserved region.

Three general memory models are presented below, but if you are converting 8080 programs to CP/M-86, you can use either the 8080 Model or Small Model and leave the Compact Model for later when your addressing needs increase. You'll use GENCMD to produce an executable program file. When you perform your conversion, ensure that you define a sufficiently large data area to address the last variable you access. You can do this in two ways. Place a "DB 0" at the end of your data area: GENCMD detects this loaded value and automatically determines the data area size. Alternatively, you can explicitly tell GENCMD the data area size using the "data" keyword with an "M" (Minimum Memory) value.

CP/M-86 itself is constructed as an 8080 Model, although you'll probably find the Small Model easier to use for your programs. This means that all the segment registers are placed at the base of CP/M-86, and your customized BIOS is identical, in most respects, to that of CP/M-80 (with changes in instruction mnemonics, of course). In fact, the only additions are found in the SETSEGB, GETSEGB, SETIOB, and GETIOB entry points in the BIOS. Your warm start subroutine is simpler since you are not required to reload the CCP and BDOS under CP/M-86. One other point: if you implement the IOBYTE facility, you'll have to define the variable in your BIOS. Taking these changes into account, you need only perform a simple translation of your CP/M-80 BIOS into 8086 code in order to implement your new BIOS.

If you've implemented CP/M-80 Version 2, you already have disk definition tables which will operate properly with CP/M-86. You may wish to attach different disk drives, or experiment with

All Information Presented Here is Proprietary to Digital Research

sector skew factors to increase performance. If so, you can use the new GENDEF utility which performs the same function as the DISKDEF macro used by MAC under CP/M-80. You'll find, however, that GENDEF provides you with more information and checks error conditions better than the DISKDEF macro.

System generation is simpler than CP/M-80, but you'll have to perform a two-step bootstrap load if you want to operate with single density floppy disks since CP/M-86 is too large to fit in the two track system area. First you load the cold start loader, and the cold start loader loads CP/M-86. With the exception of the BOOT entry point, the cold start loader BIOS, called LDBIOS, is identical to your CP/M-86 BIOS, although it can be simplified since no disk write operations take place. If you reserve enough system tracks on a non-standard disk, you can load CP/M-86 directly without the two-step boot.

BDOS entry is similar with the substitution of a software interrupt under CP/M-86 for the JMP 0005 found in CP/M-80. Although the JMP 0005 has been removed, the address field at offset 0006 is present so that programs which "size" available memory using this word value will operate without change. Entry and exit register names change, of course, but the obvious substitutions have been made. Look through the list of BDOS function numbers and you'll find that functions 0, 27, and 31 have changed slightly. Several new functions have been added, but they do not affect existing programs.

Under CP/M-80, programs terminate in one of three ways: by returning directly to the CCP, through BDOS function call 0, or by transferring control to absolute location 0000H. CP/M-86, however, supports only the first two methods of program termination. This has the side effect of not providing the automatic disk system reset following the jump to 0000H which, instead, is accomplished by entering a CONTROL-C at the CCP level.

One major philosophical difference is that in CP/M-80, all addresses sent to the BDOS are simply 16-bit values in the range 0000H to 0FFFFH. In CP/M-86, however, the addresses are really just 16-bit offsets from the DS (Data Segment) register which is set to the base of your data area. If you translate an existing CP/M-80 program to the CP/M-86 environment, your data segment will be less than 64K bytes. In this case, the DS register need not be changed following initial load, and thus all CP/M-80 addresses become simple DS-relative offsets.

You'll find many new facilities in CP/M-86 that will simplify your programming and expand your application programming capability. But, we've designed CP/M-86 to make it easy to get started: in short, if you are converting from CP/M-80 to CP/M-86, there will be no major changes beyond the translation to 8086 machine code. Further, programs you design for CP/M-86 are upward compatible with MP/M-86, our multitasking operating system, as well as CP/NET-86 which provides a distributed operating system in a network environment.

All Information Presented Here is Proprietary to Digital Research

## 2. COMMAND SETUP AND EXECUTION UNDER CP/M-86

This section discusses the operation of the Console Command Processor (CCP). The format of transient programs is discussed, along with memory models and memory image formats.

### 2.1. CCP Built-in and Transient Commands.

The operation of the CP/M-86 CCP is similar to that of CP/M-80. Upon initial cold start, the CP/M sign-on message is printed, drive A is automatically logged-in, and the standard prompt is issued at the console. CP/M-86 then waits for input command lines from the console, which may include one of the built-in commands

DIR ERA REN TYPE USER

(note that SAVE is not supported under CP/M-86 since the equivalent function is performed by DDT86).

Alternatively, the command line may begin with the name of a transient program with the assumed file type "CMD" denoting a "command file." The CMD file type differentiates transient command files used under CP/M-86 from COM files which operate under CP/M-80.

The CCP loads multiple transient programs into memory in "stack order" for execution. The transient programs themselves may load additional programs for subsequent execution. Thus, for example, a background printer spooler could first be loaded, followed by an execution of DDT86. DDT86 may, in turn, load a test program for a debugging session and transfer control to the test program between breakpoints. CP/M-86 keeps account of the order in which programs are loaded and, upon encountering a CONTROL-C, discontinues execution of the most recent program activated at the CCP level. In the above case, a CONTROL-C at the command level first discards DDT86 and its test program, while the second CONTROL-C aborts the background printer spooler. Program abort due to CONTROL-C does not reset the disk system (as is the case in CP/M-80), unless the CONTROL-C occurs at the CCP command input level.

Programs loaded by the CCP or by a transient program are given storage by CP/M-86 for their execution. The execution models are described in the following sections.

All Information Presented Here is Proprietary to Digital Research

## 2.2. Transient Program Execution Models.

The initial values of the segment registers are determined by one of three "memory models" used by the transient program, and described in the CMD file header. The three memory models are

|               |                                  |  |
|---------------|----------------------------------|--|
| 8080 Model    | Code and Data Groups Overlap     |  |
| Small Model   | Independent Code and Data Groups |  |
| Compact Model | Three or More Independent Groups |  |

The 8080 Model supports programs which are directly translated from CP/M-80 when code and data areas are intermixed. The 8080 Model consists only of a code group which, in turn, is normally a single segment of 64K bytes or less. The CS register can, however, be managed by the application program during execution so that multiple segments within the code group can be addressed.

The Small Model is similar to that defined by Intel, where the program consists of a code group and a data group. The Small Model is suitable for use by programs taken from CP/M-80 where code and data is easily separated. Note again that the code and data groups often consist of, but are not restricted to, single 64K byte segments.

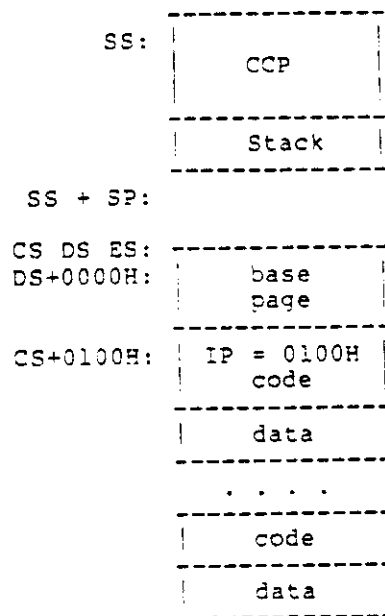
The Compact Model occurs when any of the extra, stack, or auxiliary groups are present in program. Each group may consist of one or more segments, but if any group exceeds one segment in size, or if auxiliary groups are present, then the application program must manage its own segment registers during execution in order to address all code and data areas.

The three models differ primarily in the manner in which segment registers are initialized upon transient program loading. The operating system program load function determines the memory model used by a transient program by examining the program group usage, as described below.

All Information Presented Here is Proprietary to Digital Research

### 2.3. The 8080 Memory Model.

The 8080 Model is assumed when the transient program contains only a code group. In this case, the CS, DS, and ES registers are initialized to the beginning of the code group, while the SS and SP registers remain set to a 96-byte stack area in the CCP. The Instruction Pointer Register (IP) is set to 100H, similar to CP/M-80, thus allowing base page values at the beginning of the data segment. Following program load, the 8080 Model appears as follows, where low addresses are shown at the top of the diagram:

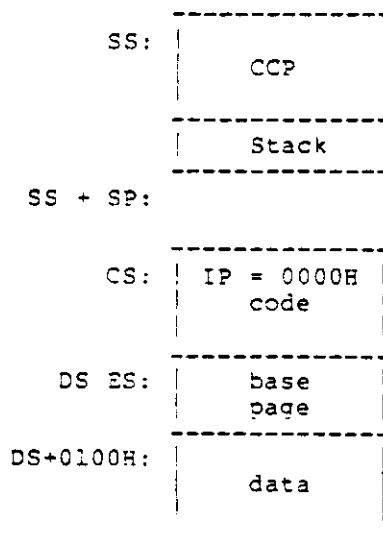


where the intermixed code and data regions are indistinguishable. The "base page" values, described below, are identical to CP/M-80, allowing simple translation from 8080, 8085, or Z80 code into the 8086 and 8088 environment.

All Information Presented Here is Proprietary to Digital Research

#### 2.4. The Small Memory Model.

The Small Model is assumed when the transient program uses both a code and data group. (In ASM86, all code is generated following a CSEG directive, while data is defined following a DSEG directive.) In this case, CS is set to the beginning of the code group, the DS and ES are set to the start of the data group, and the SS and SP registers remain in the CCP's area as shown graphically below:



where the machine code begins at CS+0000H, the "base page" values begin at DS+0000H, and the data area starts at DS+0100H.

All Information Presented Here is Proprietary to Digital Research

## 2.5. The Compact Memory Model.

The Compact Model is assumed when code and data groups are present, along with one or more of the remaining stack, extra, or auxiliary groups. In this case, the CS, DS, and ES registers are set to the base addresses of their respective areas.

It should be noted that the SS and SP registers remain in the CCP area, even if a stack group is defined. Although it may appear that the SS and SP registers should be set to address the stack group, there are two contradictions. First, the transient program may be using the stack group as a data area, in which case the Far Call could overwrite data. Second, the SS register would logically be set to the base of the group, while the SP would be set to the offset of the end of the group. However, if the stack group exceeds 64K the address range from the base to the end of the group exceeds a 16-bit offset value. Therefore, if the transient program intends to use the stack group as a stack area, the SS and SP registers must be set upon entry.

The initial configuration of the segment registers in this model is shown below:

|           |              |
|-----------|--------------|
| SS:       | CCP          |
|           | Stack        |
| SS + SP:  |              |
| CS:       | IP = 0000H   |
| CS:       | code         |
| DS:       | base<br>page |
| DS+0100H: | data         |
| ES:       | data         |

The values of the various segment registers can be programmatically changed during execution by loading from the initial values placed in base page by the CCP, as described below, thus allowing access to the entire memory space.

All Information Presented Here is Proprietary to Digital Research

## 2.6. Base Page Initialization.

Similar to CP/M-80, the base page contains default values and locations initialized by the CCP and used by the transient program. The base page occupies the regions from offset 0000H through 00FFH relative to the DS register. The values shown below in the base page for CP/M-86 include those of CP/M-80, and appear in the same relative positions.

|            |                          |     |     |
|------------|--------------------------|-----|-----|
| DS + 0000: | LC0                      | LC1 | LC2 |
| DS + 0003: | BC0                      | BC1 | M80 |
| DS + 0006: | LD0                      | LD1 | LD2 |
| DS + 0009: | BD0                      | BD1 | xxx |
| DS + 000C: | LE0                      | LE1 | LE2 |
| DS + 000F: | BE0                      | BE1 | xxx |
| DS + 0012: | LS0                      | LS1 | LS2 |
| DS + 0015: | BS0                      | BS1 | xxx |
| DS + 0018: | LX0                      | LX1 | LX2 |
| DS + 001B: | BX0                      | BX1 | xxx |
| DS + 001E: | LX0                      | LX1 | LX2 |
| DS + 0021: | BX0                      | BX1 | xxx |
| DS + 0024: | LX0                      | LX1 | LX2 |
| DS + 0027: | BX0                      | BX1 | xxx |
| DS + 002A: | LX0                      | LX1 | LX2 |
| DS + 002D: | BX0                      | BX1 | xxx |
| DS + 0030: | Not<br>Currently<br>Used |     |     |
| DS + 003B: |                          |     |     |
| DS + 005C: | Default FCB              |     |     |
| DS + 0080: | Default Buffer           |     |     |
| DS + 0100: | Begin User Data          |     |     |

where each byte is indexed by 0, 1, and 2, corresponding to the  
All Information Presented Here is Proprietary to Digital Research

standard Intel storage convention of low, middle, and high-order (most significant) byte, and "xxx" marks unused bytes. LC is the last code group location (24-bits), while BC is base paragraph address of the code group (16-bits). In the 8080 Model, the low order bytes of LC (LC0 and LC1) never exceed 0FFFFH and the high order byte (LC2) is always zero. It should be noted that bytes LD0 and LD1 appear in the same relative positions of the base page in both CP/M-80 and CP/M-86, thus easing the program translation task.

LD and BD provide the last position and paragraph base of the data group. The last position is one byte less than the group length. The M80 byte is equal to 1 when the 8080 Memory Model is in use. LE and BE provide the length and paragraph base of the optional extra group, while LS and BS give the optional stack group length and base. The bytes marked LX and BX correspond to a set of four optional independent groups which may be required for programs which execute using the Compact Model. The initial values for these descriptors are derived from the header record in the memory image file, described in the following section.

## 2.7. Transient Program Setup and Termination.

Similar to CP/M-80, the CCP parses up to two file names following the command and places the properly formatted PCB's at locations 005CH and 006CH in the base page relative to the DS register. Under CP/M-80, the default DMA address is initialized to 0080H in the base page. Due to the segmented memory of the 8086 and 8088 processors, the DMA address is divided into two parts: the DMA segment address and the DMA offset. Also, under CP/M-86, the default DMA base is set to the value of DS, and the default DMA offset is initialized to 0080H. Thus, CP/M-80 and CP/M-86 operate in the same way: both assume the default DMA address is the second half of the base page.

The CCP transfers control to the transient program through an 8086 "Far Call." In all but case A of the Compact Model, the transient program may choose to use the 96-byte CCP stack and optionally return directly to the CCP upon program termination by executing a "Far Return." Programmatic termination also occurs when BDOS function zero is executed. The operator may terminate program execution by typing a single CONTROL-C during line edited input which has the same effect as programmatic execution of BDOS function zero. Contrary to the operation of CP/M-80, no disk reset occurs and the CCP and BDOS modules are not reloaded from disk upon program termination.

### 3. COMMAND (CMD) FILE GENERATION

As mentioned previously, two utility programs are provided with CP/M-86, called GENCMD and LMCMD, which are used to produce CMD memory image files suitable for execution under CP/M-86. GENCMD accepts Intel "hex" format files as input, while LMCMD reads Intel L-module files resulting from the standard Intel LOC86 Object Code Locator utility. GENCMD is used to process output from the Digital Research ASM86 assembler and Intel's OH86 utility, while LMCMD is used when Intel compatible developmental software is available for generation of programs targeted for CP/M-86 operation.

#### 3.1. The Intel Hex File Format.

GENCMD input is in Intel "hex" format produced by both the Digital Research ASM86 assembler and the standard Intel OH86 utility program (see Intel document #9800639-03 entitled "MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users"). The CMD file produced by GENCMD is preceded by a header record which defines the memory model and memory size requirements for loading and executing the CMD file.

An Intel "hex" file consists of the traditional sequence of ASCII records in the following format:

```
-----  
:1111'a'a'a'a'tt'tt'd'd'd'd' . . . |d|c|c|  
-----
```

where the beginning of the record is marked by an ASCII colon, and each subsequent digit position contains an ASCII hexadecimal digit in the range 0-9 or A-F. The fields are defined as follows:

|      |                                                                                                                          |
|------|--------------------------------------------------------------------------------------------------------------------------|
| 11   | Record Length 00-FF (0-255 in decimal)                                                                                   |
| aaaa | Load Address, Relative to Current Segment                                                                                |
| tt   | Record Type:                                                                                                             |
|      | 00 data record, loaded starting at offset<br>aaaa from current base paragraph                                            |
|      | 01 end of file, cc = FF                                                                                                  |
|      | 02 extended address, aaaa is paragraph<br>base for subsequent data records                                               |
|      | 04 start address is aaaa (ignored, IP set<br>according to memory model in use)<br>(following are output from ASM86 only) |
|      | 81 same as 00, data belongs to code segment                                                                              |
|      | 82 same as 00, data belongs to data segment                                                                              |
|      | 83 same as 00, data belongs to stack segment                                                                             |
|      | 84 same as 00, data belongs to extra segment                                                                             |
|      | 95 paragraph address for absolute code segment                                                                           |
|      | 96 paragraph address for absolute data segment                                                                           |
|      | 87 paragraph address for absolute stack segment                                                                          |

All Information Presented Here is Proprietary to Digital Research

|    |    |                                              |
|----|----|----------------------------------------------|
|    | 88 | paragraph address for absolute extra segment |
| d  |    | Data Byte                                    |
| cc |    | Check Sum (00 - Sum of Previous Digits)      |

All characters preceding the colon for each record are ignored. (Additional hex file format information is included in the ASM86 User's Guide, along with Intel document #9800821A entitled "MCS-86 Absolute Object File Formats.")

### 3.2. Operation of GENCMD.

The GENCMD utility is invoked at the CCP level by typing

GENCMD filename parameter-list

where the filename corresponds to the hex input file with an assumed (and unspecified) file type of H86. GENCMD accepts optional parameters to specifically mark the 8080 Memory Model and to describe memory requirements of each segment group. The GENCMD parameters are listed following the filename, as shown in the command line above where the parameter-list consists of a sequence of keywords and values separated by commas or blanks. The keywords are

8080 CODE DATA EXTRA STACK X1 X2 X3 X4

The 8080 keyword forces a single code group so that the BDOS load function sets up the 8080 Memory Model for execution, thus allowing intermixed code and data within a single segment. The form of this command is

GENCMD filename 8080

The remaining keywords follow the filename or the 8080 option and define specific memory requirements for each segment group, corresponding one-to-one with the segment groups of the base page defined in the previous section. In each case, the values corresponding to each group are enclosed in square brackets and separated by commas. Each value is a hexadecimal number representing a paragraph address or segment length in paragraph units denoted by hhhh, prefixed by a single letter which defines the meaning of each value:

|       |                                                 |
|-------|-------------------------------------------------|
| Ahhhh | Load the group at absolute location hhhh        |
| Bhhhh | The group starts at hhhh in the hex file        |
| Mhhhh | The group requires a minimum of hhhh * 16 bytes |
| Xhhhh | The group requires a maximum of hhhh * 16 bytes |

Generally, the CMD file header values are derived directly from the hex file and the parameters shown above need not be included. The following situations, however, require the use of GENCMD parameters.

All Information Presented Here is Proprietary to Digital Research

The 8080 keyword is included whenever ASM86 is used in the conversion of 8080 programs to the 8086/8088 environment when code and data is intermixed within a single 64K segment, regardless of the use of CSEG and DSEG directives in the source program.

An absolute address (A value) must be given for any group which must be located at an absolute location. Normally, this value is not specified since CP/M-86 cannot generally ensure that the required memory region is available, in which case the CMD file cannot be loaded.

The B value is used when GENCMD processes files produced by Intel's OH86 utility, or similar utility program. The output from OH86 consists of a sequence of data records with no information to identify code, data, extra, stack, or auxiliary groups. In this case, the B value marks the beginning address of the group named by the keyword, causing GENCMD to load data following this address to the named group (see the examples below). Thus, the B value is normally used to mark the boundary between code and data segments when no segment information is included in the hex file. Files produced by ASM86 do not require the use of the B value since segment information is included in the hex file.

The minimum memory value (M value) is included only when the hex records do not properly define the minimum memory requirements for the named group. Generally, the code group size is determined precisely by the data records loaded into the area. That is, there is exactly one instruction byte loaded into each location of the code group and thus the size of the group is defined by the range between the lowest and highest data byte addresses. The data group, however, may contain uninitialized storage at the end of the group and thus no data records are present in the hex file which define the highest referenced data item. The highest address in the data group can be defined within the source program by including a "DB 0" as the last data item. Alternatively, the M value can be included to allocate the additional space at the end of the group. Similarly, the stack, extra, and auxiliary group sizes must be defined using the M value unless the highest addresses within the groups are implicitly defined by data records in the hex file.

The maximum memory size, given by the X value, is generally used when additional free memory is required for such purposes as I/O buffers or symbol tables. If the data area size is fixed, then the X parameter need not be included. In this case, the X value is assumed to be the same as the M value. The value XFFFF allocates the largest memory region available but, if used, the transient program must be aware that a three-byte length field is produced in

All Information Presented Here is Proprietary to Digital Research

the base page for this group where the high order byte may be non-zero. Programs converted directly from CP/M-80 should restrict this value to XFFF, producing a maximum allocation length of 0FFF0H bytes.

The following is a valid GENCMD command line

```
gencmd x code[a40] data[m30,xfff]
```

which transforms the file X.H86 into the file X.CMD with the proper header record. In this case, the code group is forced to paragraph address 40H, or equivalently, byte address 400H. The data group requires a minimum of 300H bytes, but can use up to 0FFF0H bytes, if available. Assuming a file Y.H86 exists on drive B containing Intel hex records with no interspersed segment information, the command

```
gencmd b:y data[b30,m20] extra[b50] stack[m40] x1[m40]
```

produces the file Y.CMD on drive B by selecting records beginning at address 0000H for the code segment, with records starting at 300H allocated to the data segment. The extra segment is filled from records beginning at 500H, while the stack and auxiliary segment #1 are uninitialized areas requiring a minimum of 400H bytes each. In this example, the data area requires a minimum of 200H bytes. Note again, that the B value need not be included if the Digital Research ASM86 assembler is used.

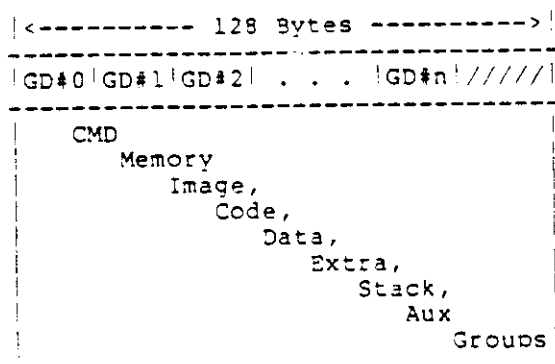
### 3.3. Operation of LMCMD.

The LMCMD utility operates in exactly the same manner as GENCMD, with the exception that LMCMD accepts an Intel L-module file as input. The primary advantage of the L-module format is that the file contains internally coded information which defines values which would otherwise be required as parameters to GENCMD, such as an absolute address of a non-relocatable segment. Currently, however, the only language processors which use this format are the standard Intel development packages, although various independent vendors will, most likely, take advantage of this format in the future.

### 3.4. Command (CMD) File Format.

The CMD file produced by GENCMD and LMCMD consists of the 128-byte header record followed immediately by the memory image. Under normal circumstances, the format of the header record is of no consequence to a programmer. For completeness, however, the various fields of this record are shown below.

All Information Presented Here is Proprietary to Digital Research



Where GD#0 through GD#n represent "Group Descriptors." Each Group Descriptor corresponds to an independently loaded region of memory, and has the following fields:

| 8-bit  | 16-bit   | 16-bit | 16-bit | 16-bit |
|--------|----------|--------|--------|--------|
| G-Form | G-Length | A-Base | G-Min  | G-Max  |

where G-Form describes the group format, or has the value zero if no more descriptors follow. If G-Form is non-zero, then the 8-bit value is decomposed into two fields:

| G-Form: |        |
|---------|--------|
| 4-bit   | 4-bit  |
| x x x x | G-Type |

The G-Type field determines the Group Descriptor type. The valid Group Descriptors have a G-Type in the range 1 through 9, as shown in the table below.

| G-Type  | Group Type                       |
|---------|----------------------------------|
| 1       | Program Code Group               |
| 2       | Independent Data Group           |
| 3       | Independent Extra Group          |
| 4       | Independent Stack Group          |
| 5       | Independent Auxiliary Group #1   |
| 6       | Independent Auxiliary Group #2   |
| 7       | Independent Auxiliary Group #3   |
| 8       | Independent Auxiliary Group #4   |
| 9       | Shared Code Group                |
| 10 - 14 | Unused, but Reserved             |
| 15      | Escape Code for Additional Types |

All Information Presented Here is Proprietary to Digital Research

All remaining values in the group descriptor are given in increments of 16-byte paragraph units with an assumed low-order 0 nibble to complete the 20-bit address. G-Length gives the number of paragraphs in the group. Given a G-length of 0080H, for example, the size of the group is 00800H = 20480 bytes. A-Base defines the base paragraph address for a non-relocatable group while G-Min and G-Max define the minimum and maximum size of the memory area to allocate to the group. G-Type 9 marks a "pure" code group for use under MP/M-86 and future versions of CP/M-86. Presently a Shared Code Group is treated as a non-shared Program Code Group under CP/M-86.

The memory model described by a header record is implicitly determined by the Group Descriptors. The 8080 Memory Model is assumed when only a code group is present, since no independent data group is named. The Small Model is implied when both a code and data group is present, but no additional group descriptors occur. Otherwise, the Compact Model is assumed when the CMD file is loaded.

All Information Presented Here is Proprietary to Digital Research

#### 4. BASIC DISK OPERATING SYSTEM FUNCTIONS

This section presents the interface conventions which allow transient program access to CP/M-86 BDOS and BIOS functions. The BDOS calls correspond closely to CP/M-80 Version 2 in order to simplify translation of existing CP/M-80 programs for operation under CP/M-86. BDOS entry and exit conditions are described first, followed by a presentation of the individual BDOS function calls.

##### 4.1. BDOS Parameters and Function Codes.

Entry to the BDOS is accomplished through the 8086 software interrupt #224, which is reserved by Intel Corporation for use by CP/M-86 and MP/M-86. The function code is passed in register CL with byte parameters in DL and word parameters in DX. Single byte values are returned in AL, word values in both AX and BX, and double word values in ES and BX. All segment registers, except ES, are saved upon entry and restored upon exit from the BDOS (corresponding to PL/M-86 conventions). The following table summarizes input and output parameter passing:

| BDOS Entry Registers | BDOS Return Registers |
|----------------------|-----------------------|
| CX Function Code     | AL Byte Value         |
| DL Byte Parameter    | AX Word Value         |
| DX Word Parameter    | BX " "                |
| DS Data Segment      | BX Double Word Offset |
|                      | ES Segment Address    |

Note that the CP/M-80 BDOS requires an "information address" as input to various functions. This address usually provides buffer or File Control Block information used in the system call. In CP/M-86, however, the information address is derived from the current DS register along with the offset given in the DX register. That is, the DX register in CP/M-86 performs the same function as the DE pair in CP/M-80, with the assumption that DS is properly set. This poses no particular problem for programs which use only a single data segment (as is the case for programs converted from CP/M-80), but when the data group exceeds a single segment, you must ensure that the DS register is set to the segment containing the data area related to the call. It should also be noted that zero values are returned for function calls which are out-of-range.

All Information Presented Here is Proprietary to Digital Research

A list of CP/M-86 calls is given below, with an asterisk following functions which differ from or are added to the set of CP/M-80 Version 2 functions.

|    |                       |     |                             |
|----|-----------------------|-----|-----------------------------|
| 0* | System Reset          | 24  | Return Login Vector         |
| 1  | Console Input         | 25  | Return Current Disk         |
| 2  | Console Output        | 26  | Set DMA Address             |
| 3  | Reader Input          | 27* | Get Addr(Alloc)             |
| 4  | Punch Output          | 28  | Write Protect Disk          |
| 5  | List Output           | 29  | Get Addr(R/O Vector)        |
| 6  | Direct Console I/O    | 30  | Set File Attributes         |
| 7  | Get I/O Byte          | 31* | Get Addr(Disk Parms)        |
| 8  | Set I/O Byte          | 32  | Set/Get User Code           |
| 9  | Print String          | 33  | Read Random                 |
| 10 | Read Console Buffer   | 34  | Write Random                |
| 11 | Get Console Status    | 35  | Compute File Size           |
| 12 | Return Version Number | 36  | Set Random Record           |
| 13 | Reset Disk System     | 37  | Reset drive                 |
| 14 | Select Disk           | 40  | Write Random with Zero Fill |
| 15 | Open File             | 50* | Direct BIOS Call            |
| 16 | Close File            | 51* | Set DMA Segment Base        |
| 17 | Search for First      | 52* | Get DMA Segment Base        |
| 18 | Search for Next       | 53* | Get Max Memory Available    |
| 19 | Delete File           | 54* | Get Max Mem at Abs Location |
| 20 | Read Sequential       | 55* | Get Memory Region           |
| 21 | Write Sequential      | 56* | Get Absolute Memory Region  |
| 22 | Make File             | 57* | Free memory region          |
| 23 | Rename File           | 58* | Free all memory             |
|    |                       | 59* | Program load                |

The individual BDOS functions are described below in three sections which cover the simple functions, file operations, and extended operations for memory management and program loading.

#### 4.2. Simple BDOS Calls.

The first set of BDOS functions cover the range 0 through 12, and perform simple functions such as a system reset, and single character I/O.

All Information Presented Here is Proprietary to Digital Research

```

*****
*                                     *
* FUNCTION 0:  SYSTEM RESET          *
*                                     *
*****
* Entry Parameters:                  *
*   Register  CL:  00H               *
*   Register  DL:  Abort code        *
*****

```

The system reset function returns control to the CP/M operating system at the CCP command level. The The abort code in DL has two possible values: if DL = 00H then the currently active program is terminated and control is returned to the CCP. If DL is a 01H, the program remains in memory and the memory allocation state remains unchanged.

```

*****
*                                     *
* FUNCTION 1:  CONSOLE INPUT         *
*                                     *
*****
* Entry Parameters:                  *
*   Register  CL:  01H               *
*                                     *
* Returned   Value:                  *
*   Register  AL:  ASCII Character   *
*****

```

The console input function reads the next character from the logical console device (CONSOLE) to register AL. Graphic characters, along with carriage return, line feed, and backspace (CONTROL-H) are echoed to the console. Tab characters (CONTROL-I) are expanded in columns of eight characters. The BDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

```

*****
*                                     *
* FUNCTION 2:  CONSOLE OUTPUT        *
*                                     *
*****
* Entry Parameters:                  *
*   Register  CL:  02H               *
*   Register  DL:  ASCII Character   *
*                                     *
*****

```

The ASCII character from DL is sent to the logical console. Tab characters expand in columns of eight characters. In addition, a check is made for start/stop scroll (CONTROL-S).

All Information Presented Here is Proprietary to Digital Research

```

*****
*                                     *
*  FUNCTION 3:  READER INPUT          *
*                                     *
*****
*  Entry Parameters:                 *
*      Register  CL:  03H             *
*                                     *
*  Returned  Value:                  *
*      Register  AL:  ASCII Character *
*****

```

The Reader Input function reads the next character from the logical reader (READER) into register AL. Control does not return until the character has been read.

```

*****
*                                     *
*  FUNCTION 4:  PUNCH OUTPUT          *
*                                     *
*****
*  Entry Parameters:                 *
*      Register  CL:  04H             *
*      Register  DL:  ASCII Character *
*****

```

The Punch Output function sends the character from register DL to the logical punch device (PUNCH).

```

*****
*                                     *
*  FUNCTION 5:  LIST OUTPUT           *
*                                     *
*****
*  Entry Parameters:                 *
*      Register  CL:  05H             *
*      Register  DL:  ASCII Character *
*****

```

The List Output function sends the ASCII character in register DL to the logical list device (LIST).

```

*****
*
*  FUNCTION 6:  DIRECT CONSOLE I/O
*
*****
*  Entry Parameters:
*      Register CL:  06H
*      Register DL:  0FFH (input) or
*                   0FEH (status) or
*                   char (output)
*
*  Returned Value:
*      Register AL:  char or status
*                   (no value)
*****

```

Direct console I/O is supported under CP/M for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M's normal control character functions (e.g., CONTROL-S and CONTROL-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under the BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register DL either contains (1) a hexadecimal FF, denoting a CONSOLE input request, or (2) a hexadecimal FE, denoting a CONSOLE status request, or (3) an ASCII character to be output to CONSOLE where CONSOLE is the logical console device. If the input value is FF, then function 6 returns AL = 00 if no character is ready. Otherwise, AL contains the next console input character. If the input value is FE, then function 6 returns AL = 00 if no character is ready and AL = FF otherwise. If the input value in DL is not FE or FF, then function 6 assumes that DL contains a valid ASCII character which is sent to the console.

```

*****
*
*  FUNCTION 7:  GET I/O BYTE
*
*****
*  Entry Parameters:
*      Register CL:  07H
*
*  Returned Value:
*      Register AL:  I/O Byte Value
*****

```

The Get I/O Byte function returns the current value of IOBYTE in register AL. The IOBYTE contains the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST provided the IOBYTE facility is implemented in the BIOS.

```

*****

```

All Information Presented Here is Proprietary to Digital Research

```
*
* FUNCTION 8: SET I/O BYTE
*
*****
* Entry Parameters:
*   Register CL: 08H
*   Register DL: I/O Byte value
*
*****
```

The Set I/O Byte function changes the system IOBYTE value to that given in register DL. This function allows transient program access to the IOBYTE in order to modify the current assignments for the logical devices CONSOLE, READER, PUNCH, and LIST.

```
*****
*
* FUNCTION 9: PRINT STRING
*
*****
* Entry Parameters:
*   Register CL: 09H
*   Register DX: String Address
*
*****
```

The Print String function sends the character string stored in memory at the location given by DX to the logical console device (CONSOLE), until a "S" is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

```

*****
*
* FUNCTION 10: READ CONSOLE BUFFER
*
*****
* Entry Parameters:
*   Register CL: 0AH
*   Register DX: Buffer Address
*
* Returned Value:
*   Console Characters in Buffer
*****

```

The Read Buffer function reads a line of edited console input into a buffer addressed by register DX from the logical console device (CONSOLE). Console input is terminated when either the input buffer is filled or when a return (CONTROL-M) or a line feed (CONTROL-J) character is entered. The input buffer addressed by DX takes the form:

```

DX: +0 +1 +2 +3 +4 +5 +6 +7 +8 . . . +n
-----
|mx|nc|c1|c2|c3|c4|c5|c6|c7| . . . |??|
-----

```

where "mx" is the maximum number of characters which the buffer will hold, and "nc" is the number of characters placed in the buffer. The characters entered by the operator follow the "nc" value. The value "mx" must be set prior to making a function 10 call and may range in value from 1 to 255. Setting mx to zero is equivalent to setting mx to one. The value "nc" is returned to the user and may range from 0 to mx. If nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. Note that a terminating return or line feed character is not placed in the buffer and not included in the count "nc".

A number of editing control functions are supported during console input under function 10.

|           |                                         |
|-----------|-----------------------------------------|
| rub/del   | removes and echoes the last character   |
| CONTROL-C | reboots when at the beginning of line   |
| CONTROL-E | causes physical end of line             |
| CONTROL-H | backspaces one character position       |
| CONTROL-J | (line feed) terminates input line       |
| CONTROL-M | (return) terminates input line          |
| CONTROL-R | retypes the current line after new line |
| CONTROL-U | removes current line after new line     |
| CONTROL-X | backspaces to beginning of current line |

Certain functions which return the carriage to the leftmost position (e.g., control-X) do so only to the column position where the prompt ended (in earlier releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.

All Information Presented Here is Proprietary to Digital Research

```

*****
*
*  FUNCTION 11: GET CONSOLE STATUS
*
*****
*  Entry Parameters:
*    Register CL: 0BH
*
*  Returned Value:
*    Register AL: Console Status
*****

```

The Console Status function checks to see if a character has been typed at the logical console device (CONSOLE). If a character is ready, the value 01H is returned in register AL. Otherwise a 00H value is returned.

```

*****
*
*  FUNCTION 12: RETURN VERSION NUMBER
*
*****
*  Entry Parameters:
*    Register CL: 0CH
*
*  Returned Value:
*    Register BX: Version Number
*****

```

Function 12 provides information which allows version independent programming. A two-byte value is returned, with BH = 00 designating the CP/M release (BH = 01 for MP/M), and BL = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register BL, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. In order to provide version number compatibility, the initial release of CP/M-86 starts with 2.2.

#### 4.3. BDOS File Operations.

Functions 12 through 52 are related to disk file operations under CP/M-86. In many of these operations, DX provides the DS-relative offset to a file control block (FCB). The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, or a sequence of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at offset 005CH from the DS register can be used for random access files, since bytes 007DH, 007EH, and 007FH are available for this purpose. The FCB format is shown below with the following fields:

All Information Presented Here is Proprietary to Digital Research

```

-----
|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|r1|r2|
-----
00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35

```

where

dr        drive code (0 - 16)  
           0 => use default drive for file  
           1 => auto disk select drive A,  
           2 => auto disk select drive B,  
           ...  
           16=> auto disk select drive P.

f1...f8    contain the file name in ASCII  
           upper case, with high bit = 0

t1,t2,t3   contain the file type in ASCII  
           upper case, with high bit = 0  
           t1', t2', and t3' denote the  
           bit of these positions,  
           t1' = 1 => Read/Only file,  
           t2' = 1 => SYS file, no DIR list

ex        contains the current extent number,  
           normally set to 00 by the user, but  
           in range 0 - 31 during file I/O

s1        reserved for internal system use

s2        reserved for internal system use, set  
           to zero on call to OPEN, MAKE, SEARCH

rc        record count for extent "ex,"  
           takes on values from 0 - 128

d0...dn   filled-in by CP/M, reserved for  
           system use

cr        current record to read or write in  
           a sequential file operation, normally  
           set to zero by user

r0,r1,r2   optional random record number in the  
           range 0-65535, with overflow to r2,  
           r0,r1 constitute a 16-bit value with  
           low byte r0, and high byte r1

For users of earlier versions of CP/M, it should be noted in passing that both CP/M-80 Version 2 and CP/M-86 perform directory operations in a reserved area of memory which does not affect write buffer content, except in the case of Search and Search Next where the directory record is copied to the current DMA address.

All Information Presented Here is Proprietary to Digital Research

There are three error situations which the BDOS may encounter during file processing, initiated as a result of a BDOS File I/O function call. When one of these conditions is detected, the BDOS issues the following message to the console:

BDOS ERR ON x: error

where x is the drive name of the drive selected when the error condition is detected, and "error" is one of the three messages:

BAD SECTOR      SELECT      R/O

These error situations are trapped by the BDOS, and thus the executing transient program is temporarily halted when the error is detected. No indication of the error situation is returned to the transient program.

The "BAD SECTOR" error is issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes BIOS sector read and write commands as part of the execution of BDOS file related system calls. If the BIOS read or write routine detects a hardware error, it returns an error code to the BDOS resulting in this error message. The operator may respond to this error in two ways: a CONTROL-C terminates the executing program, while a RETURN instructs CP/M-86 to ignore the error and allow the program to continue execution.

The "SELECT" error is also issued as the result of an error condition returned to the BDOS from the BIOS module. The BDOS makes a BIOS disk select call prior to issuing any BIOS read or write to a particular drive. If the selected drive is not supported in the BIOS module, it returns an error code to the BDOS resulting in this error message. CP/M-86 terminates the currently running program and returns to the command level of the CCP following any input from the console.

The "R/O" message occurs when the BDOS receives a command to write to a drive that is in read-only status. Drives may be placed in read-only status explicitly as the result of a STAT command or BDOS function call, or implicitly if the BDOS detects that disk media has been changed without performing a "warm start." The ability to detect changed media is optionally included in the BIOS, and exists only if a checksum vector is included for the selected drive. Upon entry of any character at the keyboard, the transient program is aborted, and control returns to the CCP.

All Information Presented Here is Proprietary to Digital Research

```

*****
*                                     *
*  FUNCTION 13: RESET DISK SYSTEM    *
*                                     *
*****
*  Entry Parameters:                *
*      Register CL: 0DH              *
*                                     *
*****

```

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected. This function can be used, for example, by an application program which requires disk changes during operation. Function 37 (Reset Drive) can also be used for this purpose.

```

*****
*                                     *
*  FUNCTION 14: SELECT DISK          *
*                                     *
*****
*  Entry Parameters:                *
*      Register CL: 0EH              *
*      Register DL: Selected Disk    *
*                                     *
*****

```

The Select Disk function designates the disk drive named in register DL as the default disk for subsequent file operations, with DL = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. In addition, the designated drive is logged-in if it is currently in the reset state. Logging-in a drive places it in "on-line" status which activates the drive's directory until the next cold start, warm start, disk system reset, or drive reset operation. FCB's which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

All Information Presented Here is Proprietary to Digital Research

```
*****
*
* FUNCTION 15: OPEN FILE
*
*****
* Entry Parameters:
*   Register CL: 0FH
*   Register DX: FCB Address
*
* Returned Value:
*   Register AL: Directory Code
*****
```

The Open File operation is used to activate a FCB specifying a file which currently exists in the disk directory for the currently active user number. The BDOS scans the disk directory of the drive specified by byte 0 of the FCB referenced by DX for a match in positions 1 through 12 of the referenced FCB, where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, byte "ex" of the FCB is set to zero before making the open call.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a successful open operation is completed. Further, an FCB not activated by either an open or make function must not be used in BDOS read or write commands. Upon return, the open function returns a "directory code" with the value 0 through 3 if the open was successful, or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

All Information Presented Here is Proprietary to Digital Research

```
*****
*                                     *
* FUNCTION 16: CLOSE FILE           *
*                                     *
*****
* Entry Parameters:                 *
*   Register CL: 10H                 *
*   Register DX: FCB Address         *
*                                     *
* Returned Value:                   *
*   Register AL: Directory Code      *
*****
```

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DX has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

```

*****
*                                     *
*  FUNCTION 17: SEARCH FOR FIRST      *
*                                     *
*****
*  Entry Parameters:                 *
*    Register CL:  11H                *
*    Register DX:  FCB Address        *
*                                     *
*  Returned Value:                   *
*    Register AL:  Directory Code     *
*****

```

Search First scans the directory for a match with the file given by the FCB addressed by DX. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is  $A * 32$  (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "fl" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.

```

*****
*                                     *
*  FUNCTION 18: SEARCH FOR NEXT      *
*                                     *
*****
*  Entry Parameters:                 *
*    Register CL:  12H                *
*                                     *
*  Returned Value:                   *
*    Register AL:  Directory Code     *
*****

```

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match. In terms of execution sequence, a function 18 call must follow either a function 17 or function 18 call with no other intervening BDOS disk related function calls.

All Information Presented Here is Proprietary to Digital Research

```
*****
*
* FUNCTION 19: DELETE FILE
*
*****
* Entry Parameters:
*   Register CL: 13H
*   Register DX: FCB Address
*
* Returned Value:
*   Register AL: Return Code
*****
```

The Delete File function removes files which match the FCB addressed by DX. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions. Function 19 returns a 0FFH (decimal 255) if the referenced file or files cannot be found, otherwise a value of zero is returned.

```
*****
*
* FUNCTION 20: READ SEQUENTIAL
*
*****
* Entry Parameters:
*   Register CL: 14H
*   Register DX: FCB Address
*
* Returned Value:
*   Register AL: Return Code
*****
```

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128 byte record from the file into memory at the current DMA address. The record is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The "cr" field must be set to zero following the open call by the user if the intent is to read sequentially from the beginning of the file. The value 00H is returned in the AL register if the read operation was successful, while a value of 01H is returned if no data exists at the next record position of the file. Normally, the no data situation is encountered at the end of a file. However, it can also occur if an attempt is made to read a data block which has not been previously written, or an extent which has not been created. These situations are usually restricted to files created or appended by use of the BDOS Write Random command (function 34).

All Information Presented Here is Proprietary to Digital Research

```

*****
*
* FUNCTION 21: WRITE SEQUENTIAL
*
*****
* Entry Parameters:
*   Register CL: 15H
*   Register DX: FCB Address
*
* Returned Value:
*   Register AL: Return Code
*****

```

Given that the FCB addressed by DX has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. the record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. The "cr" field must be set to zero following an open or make call by the user if the intent is to write sequentially from the beginning of the file. Register AL = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to one of the following conditions:

- 01 No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
- 02 No available data block - This condition is encountered when the write command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.

All Information Presented Here is Proprietary to Digital Research

```

*****
*                                     *
* FUNCTION 22: MAKE FILE             *
*                                     *
*****
* Entry Parameters:                  *
*   Register CL: 16H                 *
*   Register DX: FCB Address         *
*                                     *
* Returned Value:                    *
*   Register AL: Directory Code      *
*****

```

The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The BDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

```

*****
*                                     *
* FUNCTION 23: RENAME FILE           *
*                                     *
*****
* Entry Parameters:                  *
*   Register CL: 17H                 *
*   Register DX: FCB Address         *
*                                     *
* Returned Value:                    *
*   Register AL: Return Code         *
*****

```

The Rename function uses the FCB addressed by DX to change all directory entries of the file specified by the file name in the first 16 bytes of the FCB to the file name in the second 16 bytes. It is the user's responsibility to insure that the file names specified are valid CP/M unambiguous file names. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is ignored. Upon return, register AL is set to a value of zero if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

All Information Presented Here is Proprietary to Digital Research

```
*****
*
* FUNCTION 24: RETURN LOGIN VECTOR
*
*****
* Entry Parameters:
*   Register CL: 18H
*
* Returned Value:
*   Register BX: Login Vector
*****
```

The login vector value returned by CP/M is a 16-bit value in BX, where the least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field.

```
*****
*
* FUNCTION 25: RETURN CURRENT DISK
*
*****
* Entry Parameters:
*   Register CL: 19H
*
* Returned Value:
*   Register AL: Current Disk
*****
```

Function 25 returns the currently selected default disk number in register AL. The disk numbers range from 0 through 15 corresponding to drives A through P.

All Information Presented Here is Proprietary to Digital Research

```

*****
*
*  FUNCTION 26: SET DMA ADDRESS
*
*****
*  Entry Parameters:
*      Register CL: 1AH
*      Register DX: DMA Offset
*
*****

```

"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128 byte data record resides before a disk write and after a disk read. In the CP/M-86 environment, the Set DMA function is used to specify the offset of the read or write buffer from the current DMA base. Therefore, to specify the DMA address, both a function 26 call and a function 51 call is required. Thus, the DMA address becomes the value specified by DX plus the DMA base value until it is changed by a subsequent Set DMA function.

```

*****
*
*  FUNCTION 27: GET ADDR(ALLOC)
*
*****
*  Entry Parameters:
*      Register CL: 1BH
*
*  Returned Value:
*      Register BX: ALLOC Offset
*      Register ES: Segment base
*
*****

```

An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the segment base and the offset address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only.

All Information Presented Here is Proprietary to Digital Research

```

*****
*
*  FUNCTION 28: WRITE PROTECT DISK
*
*****
*  Entry Parameters:
*    Register CL: 1CH
*
*****

```

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold start, warm start, disk system reset, or drive reset operation produces the message:

Bdos Err on d: R/O

```

*****
*
*  FUNCTION 29: GET READ/ONLY VECTOR
*
*****
*  Entry Parameters:
*    Register CL: 1DH
*
*  Returned Value:
*    Register BX: R/O Vector Value*
*****

```

Function 29 returns a bit vector in register BX which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive AL, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

```
*****
*
*  FUNCTION 30: SET FILE ATTRIBUTES
*
*****
*  Entry Parameters:
*      Register CL: 1EH
*      Register DX: FCB Address
*
*  Returned Value:
*      Register AL: Return Code
*****
```

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O, System and Archive attributes (t1', t2', and t3') can be set or reset. The DX pair addresses a FCB containing a file name with the appropriate attributes set or reset. It is the user's responsibility to insure that an unambiguous file name is not specified. Function 30 searches the FCB specified or default disk drive directory area for directory entries that belong to the current user number and that match the FCB specified name and type fields. All matching directory entries are updated to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' are reserved for future system expansion. The currently assigned attributes are defined as follows:

- t1': The R/O attribute indicates if set that the file is in read/only status. BDOS will not allow write commands to be issued to files in R/O status.
- t2': The System attribute is referenced by the CP/M DIR utility. If set, DIR will not display the file in a directory display.
- t3': The Archive attribute is reserved but not actually used by CP/M. If set it indicates that the file has been written to back up storage by a user written archive program. To implement this facility, the archive program sets this attribute when it copies a file to back up storage; any programs updating or creating files reset this attribute. Further, the archive program backs up only those files that have the Archive attribute reset. Thus, an automatic back up facility restricted to modified files can be easily implemented.

Function 30 returns with register AL set to 0FFH (255 decimal) if the referenced file cannot be found, otherwise a value of zero is returned.

All Information Presented Here is Proprietary to Digital Research

```

*****
*
* FUNCTION 31: GET ADDR(DISK PARMS)
*
*****
* Entry Parameters:
*   Register CL: 1FH
*
* Returned Value:
*   Register BX: DPB Offset
*   Register ES: Segment Base
*****

```

The offset and the segment base of the BIOS resident disk parameter block of the currently selected drive is returned in BX and ES as a result of this function call. This control block can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility. Refer to Section 6.3 for the definition of the BIOS disk parameter block.

```

*****
*
* FUNCTION 32: SET/GET USER CODE
*
*****
* Entry Parameters:
*   Register CL: 20H
*   Register DL: 0FFH (get) or
*               User Code (set)
*
* Returned Value:
*   Register AL: Current Code or
*               no value
*****

```

An application program can change or interrogate the currently active user number by calling function 32. If register DL = 0FFH, then the value of the current user number is returned in register AL, where the value is in the range 0 to 15. If register DL is not 0FFH, then the current user number is changed to the value of DL (modulo 16).

All Information Presented Here is Proprietary to Digital Research

```

*****
*
*  FUNCTION 33: READ RANDOM
*
*****
*  Entry Parameters:
*      Register CL: 21H
*      Register DX: FCB Address
*
*  Returned Value:
*      Register AL: Return Code
*****

```

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of any size file. In order to access a file using the Read Random function, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the FCB is properly initialized for subsequent random access operations. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

All Information Presented Here is Proprietary to Digital Research

Error codes returned in register AL following a random read are listed below.

- 01 Reading unwritten data - This error code is returned when a random read operation accesses a data block which has not been previously written.
- 02 (not returned by the Random Read command)
- 03 Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a read random operation on an FCB that has not been opened.
- 04 Seek to unwritten extent - This error code is returned when a random read operation accesses an extent that has not been created. This error situation is equivalent to error 01.
- 05 (not returned by the Random Read command)
- 06 Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.

Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

```
*****
*                                     *
*  FUNCTION 34: WRITE RANDOM         *
*                                     *
*****
*  Entry Parameters:                *
*    Register CL: 22H               *
*    Register DX: FCB Address       *
*                                     *
*  Returned Value:                  *
*    Register AL: Return Code       *
*****
```

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to

All Information Presented Here is Proprietary to Digital Research

correspond to the random record which is being written. Sequential read or write operations can commence following a random write, with the note that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. In particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

In order to access a file using the Write Random function, the base extent (extent 0) must first be opened. As in the Read Random function, this ensures that the FCB is properly initialized for subsequent random access operations. If the file is empty, a Make File function must be issued for the base extent. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests.

Upon return from a Write Random call, register AL either contains an error code, as listed below, or the value 00 indicating the operation was successful.

- 01 (not returned by the Random Write command)
- 02 No available data block - This condition is encountered when the Write Random command attempts to allocate a new data block to the file and no unallocated data blocks exist on the selected disk drive.
- 03 Cannot close current extent - This error code is returned when BDOS cannot close the current extent prior to moving to the new extent containing the record specified by bytes r0,r1 of the FCB. This error can be caused by an overwritten FCB or a write random operation on an FCB that has not been opened.
- 04 (not returned by the Random Write command)
- 05 No available directory space - This condition occurs when the write command attempts to create a new extent that requires a new directory entry and no available directory entries exist on the selected disk drive.
- 06 Random record number out of range - This error code is returned whenever byte r2 of the FCB is non-zero.

```

*****
*
*  FUNCTION 35: COMPUTE FILE SIZE
*
*****
*  Entry Parameters:
*      Register CL:  23H
*      Register DX:  FCB Address
*
*  Returned Value:
*      Random Record Field Set
*****

```

When computing the size of a file, the DX register addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, a single record with record number 65535 (CP/M's maximum record number) is written to a file using the Write Random function, then the virtual size of the file is 65536 records, although only one block of data is actually allocated.

All Information Presented Here is Proprietary to Digital Research

```

*****
*
*  FUNCTION 36: SET RANDOM RECORD
*
*****
*  Entry Parameters:
*      Register CL: 24H
*      Register DX: FCB Address
*
*  Returned Value:
*      Random Record Field Set
*****

```

The Set Random Record function causes the BDOS to automatically produce the random record position of the next record to be accessed from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position minus one is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the next record in the file.

All Information Presented Here is Proprietary to Digital Research

```
*****
*
* FUNCTION 37: RESET DRIVE
*
*****
* Entry Parameters:
*   Register CL: 25H
*   Register DX: Drive Vector
*
* Returned Value :
*   AL: 00H
*****
```

The Reset Drive function is used to programmatically restore specified drives to the reset state (a reset drive is not logged-in and is in read/write status). The passed parameter in register DX is a 16 bit vector of drives to be reset, where the least significant bit corresponds to the first drive A, and the high order bit corresponds to the sixteenth drive, labelled P. Bit values of "1" indicate that the specified drive is to be reset.

In order to maintain compatibility with MP/M, CP/M returns a zero value for this function.

```
*****
*
* FUNCTION 40: WRITE RANDOM WITH
*              ZERO FILL
*
*****
* Entry Parameters:
*   Register CL: 28H
*   Register DX: FCB address
*
* Returned Value:
*   Register AL: Return Code
*****
```

The Write Random With Zero Fill function is similar to the Write Random function (function 34) with the exception that a previously unallocated data block is initialized to records filled with zeros before the record is written. If this function has been used to create a file, records accessed by a read random operation that contain all zeros identify unwritten random record numbers. Unwritten random records in allocated data blocks of files created using the Write Random function contain uninitialized data.

All Information Presented Here is Proprietary to Digital Research

```

*****
*                                     *
* FUNCTION 50:  DIRECT BIOS CALL    *
*                                     *
*****
* Entry Parameters:                 *
*   Register  CL:  32H              *
*   Register  DX:  BIOS Descriptor *
*****

```

Function 50 provides a direct BIOS call and transfers control through the BDOS to the BIOS. The DX register addresses a five-byte memory area containing the BIOS call parameters:

| 8-bit | 16-bit    | 16-bit    |
|-------|-----------|-----------|
| ----- |           |           |
| Func  | value(CX) | value(DX) |
| ----- |           |           |

where Func is the BIOS function number, starting at zero, and value(CX) and value(DX) are the 16-bit values which would normally be passed directly in the CX and DX registers with the BIOS call. The CX and DX values are loaded into the 8086 registers before the BIOS call is initiated.

```

*****
*                                     *
* FUNCTION 51:  SET DMA BASE        *
*                                     *
*****
* Entry Parameters:                 *
*   Register  CL:  33H              *
*   Register  DX:  Base Address    *
*****

```

Function 51 sets the base register for subsequent DMA transfers. The word parameter in DX is a paragraph address and is used with the DMA offset to specify the address of a 128 byte buffer area to be used in the disk read and write functions. Note that upon initial program loading, the default segment base is set to address of the user's data segment (the initial value of DS) and the DMA offset is set to 0080H, which provides access to the default buffer in the base page.

```

*****
*
* FUNCTION 52: GET DMA BASE
*
*****
* Entry Parameters:
*   Register   CL: 34H
*
* Returned Value:
*   Register   BX: DMA Offset
*   Register   ES: DMA Segment
*****

```

Function 52 returns the current DMA Base Segment address in ES, with the current DMA Offset in DX.

#### 4.4. BDOS Memory Management and Load Operations.

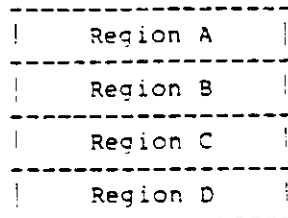
Memory is allocated in two distinct ways under CP/M-86. The first is through a static allocation map, located within the BIOS, that defines the physical memory which is available on the host system. In this way, it is possible to operate CP/M-86 in a memory configuration which is a mixture of up to eight non-contiguous areas of RAM or ROM, along with reserved, missing, or faulty memory regions. In a simple RAM-based system with contiguous memory, the static map defines a single region, usually starting at the end of the BIOS and extending up to the end of available memory.

Once memory is physically mapped in this manner, CP/M-86 performs the second level of dynamic allocation to support transient program loading and execution. CP/M-86 allows dynamic allocation of memory into, again, eight regions. A request for allocation takes place either implicitly, through a program load operation, or explicitly through the BDOS calls given in this section. Programs themselves are loaded in two ways: through a command entered at the CCP level, or through the BDOS Program Load operation (function 59). Multiple programs can be loaded at the CCP level, as long as each program executes a System Reset (function 0) and remains in memory (DL = 01H). Multiple programs of this sort only receive control by intercepting interrupts, and thus under normal circumstances there is only one transient program in memory at any given time. If, however, multiple programs are present in memory then CONTROL-C characters entered by the operator delete these programs in the opposite order in which they were loaded no matter which program is actively reading the console.

Any given program loaded through a CCP command can, itself, load additional programs and allocate data areas. Suppose four regions of memory are allocated in the following order: a program is loaded at the CCP level through an operator command. The CMD file header is read, and the entire memory image consisting of the program and its data is loaded into region A, and execution begins.

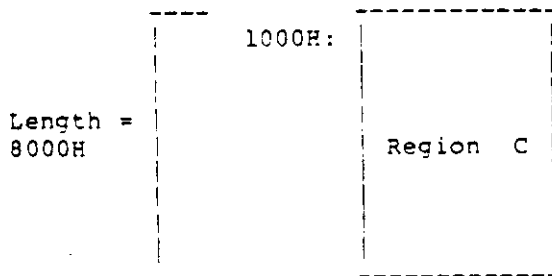
All Information Presented Here is Proprietary to Digital Research

This program, in turn, calls the BDOS Program Load function (59) to load another program into region B, and transfers control to the loaded program. The region B program then allocates an additional region C, followed by a region D. The order of allocation is shown graphically below:

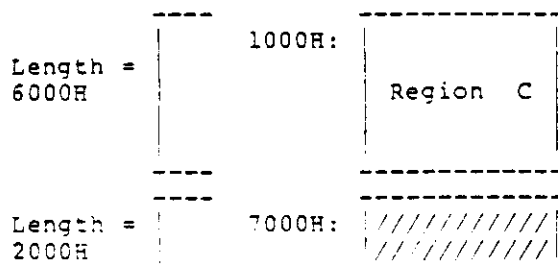


There is a hierarchical ownership of these regions: the program in A controls all memory from A through D. The program in B also controls regions B through D. The program in A can release regions B through D, if desired, and reload yet another program. DDT86, for example, operates in this manner by executing the Free Memory call (function 57) to release the current program before loading another test program. Further, the program in B can release regions C and D if required by the application. It must be noted, however, that if either program terminates by reading a CONTROL-C or executing a System Reset (with DL = 00H) then all four regions A through D are released.

A transient program may release a portion of a region, allowing the released portion to be assigned on the next allocation request. The released portion must, however, be at the beginning or end of the region. Suppose, for example, the program in region B above receives 800H paragraphs at paragraph location 100H following its first allocation request:



Suppose further that region D is then allocated. The last 200H paragraphs in region C can be returned without affecting region D by releasing the 200H paragraphs beginning at paragraph base 700H, resulting in



The region beginning at paragraph address 700H is now available for allocation in the next request. Again, note that memory requests will fail if more than eight simultaneous allocated regions are in effect.

Memory management functions beginning at 53 reference a Memory Control Block (MCB), defined in the calling program, which takes the form:

|      | 16-bit | 16-bit   | 8-bit |
|------|--------|----------|-------|
| MCB: | M-Base | M-Length | M-Ext |

where M-Base and M-Length are either input or output values expressed in 16-byte paragraph units, and M-Ext is a returned byte value, as defined specifically with each function code. An error condition is normally flagged with a 0FFH returned value in order to match the file error conventions of CP/M.

```
*****
*                                     *
*  FUNCTION 53:  GET MAX MEM          *
*                                     *
*  Entry Parameters:                  *
*    Register CL:  35H                *
*    Register DX:  Offset of MCB      *
*                                     *
*  Returned Value:                    *
*    Register AL:  Return Code        *
*****
```

Function 53 finds the largest available memory region which is less than or equal to M-Length paragraphs. If successful, M-Base is set to the base paragraph address of the available area, and M-Length to the paragraph length. AL has the value 0FFH upon return if the no memory is available, and 00H if the request was

All Information Presented Here is Proprietary to Digital Research

successful. M-Ext is set to 1 if there is additional memory for allocation, and 0 if no additional memory is available.

```

*****
*
* FUNCTION 54: GET ABS MAX
*
*****
* Entry Parameters:
*   Register CL: 36H
*   Register DX: Offset of MCB
*
* Returned Value:
*   Register AL: Return Code
*****

```

Function 54 is used to allocate the largest possible region at the absolute paragraph boundary given by M-Base, for a maximum of M-Length paragraphs. M-Length is set to the actual length if successful. AL has the value 0FFH upon return if no memory is available at the absolute address, and 00H if the request was successful.

```

*****
*
* FUNCTION 55: ALLOC MEM
*
*****
* Entry Parameters:
*   Register CL: 37H
*   Register DX: Offset of MCB
*
* Returned Value:
*   Register AL: Return Code
*****

```

The allocate memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length. Function 55 returns in the user's MCB the base paragraph address of the allocated region. Register AL contains a 00H if the request was successful and a 0FFH if the memory could not be allocated.

```

*****
*                                     *
* FUNCTION 56:  ALLOC ABS MEM        *
*                                     *
*****
* Entry Parameters:                  *
*   Register CL:  38H                *
*   Register DX:  Offset of MCB      *
*                                     *
* Returned Value:                    *
*   Register AL:  Return Code        *
*****

```

The allocate absolute memory function allocates a memory area according to the MCB addressed by DX. The allocation request size is obtained from M-Length and the absolute base address from M-Base. Register AL contains a 00H if the request was successful and a 0ffH if the memory could not be allocated.

```

*****
*                                     *
* FUNCTION 57:  FREE MEM             *
*                                     *
*****
* Entry Parameters:                  *
*   Register CL:  39H                *
*   Register DX:  Offset of MCB      *
*****

```

Function 57 is used to release memory areas allocated to the program. The value of the M-Ext field controls the operation of this function: if M-Ext = 0FFH then all memory areas allocated by the calling program are released. Otherwise, the memory area of length M-Length at location M-Base given in the MCB addressed by DX is released (the M-Ext field should be set to 00H in this case). As described above, either an entire allocated region must be released, or the entire end of a region must be released: the middle section cannot be returned under CP/M-86.

All Information Presented Here is Proprietary to Digital Research

```
*****
*
* FUNCTION 58:  FREE ALL MEM
*
*****
* Entry Parameters:
*   Register CL:  3AH
*****
```

Function 58 is used to release all memory in the CP/M-86 environment (normally used only by the CCP upon initialization).

```
*****
*
* FUNCTION 59:  PROGRAM LOAD
*
*****
* Entry Parameters:
*   Register CL:  3BH
*   Register DX:  Offset of FCB
*
* Returned Value:
*   Register BX:  Base Page Addr
*****
```

Function 59 loads a CMD file. Upon entry, register DX contains the DS relative offset of a successfully opened FCB which names the input CMD file. AX has the value 0FFFFH if the program load was unsuccessful. Otherwise, AX and BX both contain the paragraph address of the base page belonging to the loaded program. The base address and segment length of each segment is stored in the base page. Note that upon program load at the CCP level, the DMA base address is initialized to the base page of the loaded program, and the DMA offset address is initialized to 0080H. However, this is a function of the CCP, and a function 59 does not establish a default DMA address. It is the responsibility of the program which executes function 59 to execute function 51 to set the DMA base and function 26 to set the DMA offset before passing control to the loaded program.

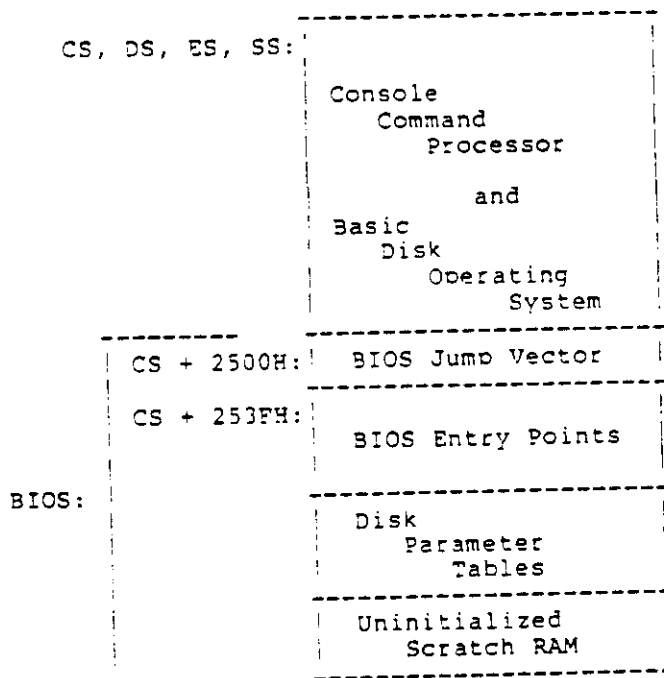
All Information Presented Here is Proprietary to Digital Research

## 5. BASIC I/O SYSTEM (BIOS) ORGANIZATION

The distribution version of CP/M-86 is setup for operation with the Intel SBC 86/12 microprocessor and an Intel 204 diskette controller. All hardware dependencies are, however, concentrated in subroutines which are collectively referred to as the Basic I/O System, or BIOS. A CP/M-86 system implementor can modify these subroutines, as described below, to tailor CP/M-86 to fit nearly any 8086 or 8088 operating environment. This section describes the actions of each BIOS entry point, and defines variables and tables referenced within the BIOS. The discussion of Disk Definition Tables is, however, treated separately in the next section of this manual.

### 5.1. Organization of the BIOS.

The BIOS portion of CP/M-86 resides in the topmost portion of the operating system (highest addresses), and take the following general form:



As described in the following sections, the CCP and BDOS are supplied with CP/M-86 in hex file form as CPM.H86. In order to implement CP/M-86 on non-standard hardware, you must create a BIOS which performs the functions listed below and concatenate the result to the end of the CPM.H86 file. The GENCMD utility is then

All Information Presented Here is Proprietary to Digital Research

used to produce the CPM.SYS file for subsequent load by the cold start loader. The cold start loader itself contains a simplified form of the BIOS, called the LBIOS (Loader BIOS), which loads the resulting CPM.SYS file into memory at the desired location (usually 0400H). The procedure to follow in construction and execution of the cold start loader and the CP/M-86 Loader is given in a later section.

Appendix D contains a listing of the standard CP/M-86 BIOS for the Intel SBC 86/12 system using the Intel 204 Controller Board. Appendix E shows a sample "skeletal" BIOS that contains the essential elements with the device drivers removed. You may wish to review these listings in order to determine the overall structure of the BIOS.

## 5.2. The BIOS Jump Vector.

Entry to the BIOS is through a "jump vector" located at offset 2500H from the base of the operating system, as shown graphically above, and as listed below. The jump vector is a sequence of 21 three-byte jump instructions which transfer program control to the individual BIOS entry points. Although some non-essential BIOS subroutines may contain a single return (RET) instruction, the corresponding jump vector element must be present in the order shown below.

Parameters for the individual subroutines in the BIOS are passed in the CX and DX registers, when required. CX receives the first parameter, DX is used for a second argument. Return values are passed in the registers according to type: Byte values are returned in AL. Word values (16 bits) are returned in BX. Specific parameters and returned values are described with each subroutine. The BIOS jump vector takes the following form:

All Information Presented Here is Proprietary to Digital Research

```

2500H    JMP BOOT           ; ARRIVE HERE FROM COLD BOOT
2503H    JMP WBOOT         ; ARRIVE HERE FOR WARM START
2506H    JMP CONST         ; CHECK FOR CONSOLE CHAR READY
2509H    JMP CONIN         ; READ CONSOLE CHARACTER IN
250CH    JMP CONOUT        ; WRITE CONSOLE CHARACTER OUT
250FH    JMP LIST          ; WRITE LISTING CHARACTER OUT
2512H    JMP PUNCH         ; WRITE CHAR TO PUNCH DEVICE
2515H    JMP READER        ; READ READER DEVICE
2518H    JMP HOME          ; MOVE TO TRACK 00
251BH    JMP SELDSK        ; SELECT DISK DRIVE
251EH    JMP SETTRK        ; SET TRACK NUMBER
2521H    JMP SETSEC        ; SET SECTOR NUMBER
2524H    JMP SETDMA        ; SET DMA OFFSET ADDRESS
2527H    JMP READ          ; READ SELECTED SECTOR
252AH    JMP WRITE         ; WRITE SELECTED SECTOR
252DH    JMP LISTST        ; RETURN LIST STATUS
2530H    JMP SECTTRAN      ; SECTOR TRANSLATE
2533H    JMP SETDMAB       ; SET DMA SEGMENT ADDRESS
2536H    JMP GETSEGB       ; GET MEM DESC TABLE OFFSET
2539H    JMP GETIOB        ; GET I/O MAPPING BYTE
253CH    JMP SETIOB        ; SET I/O MAPPING BYTE

```

There are three major divisions in the BIOS jump table: system (re)initialization subroutines, simple character I/O subroutines, and disk I/O subroutines.

### 5.3. Simple Peripheral Devices.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices, and are assigned to physical devices within the BIOS. Device characteristics are defined below.

|         |                                                                                                                                                                                  |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONSOLE | The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype. |
| LIST    | The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.                                                   |
| PUNCH   | The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.                                                                   |
| READER  | The principal tape reading device, such as a simple optical reader or Teletype.                                                                                                  |

All Information Presented Here is Proprietary to Digital Research

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by you may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other transient program. Alternately, the PUNCH and LIST subroutines can just simply return, and the READER subroutine can return with a LAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, you can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in the BIOS is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

|        | most significant |          | least significant |          |
|--------|------------------|----------|-------------------|----------|
| IOBYTE | LIST             | PUNCH    | READER            | CONSOLE  |
|        | bits 6,7         | bits 4,5 | bits 2,3          | bits 0,1 |

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below

#### CONSOLE field (bits 0,1)

- 0 - console is assigned to the console printer (TTY:)
- 1 - console is assigned to the CRT device (CRT:)
- 2 - batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
- 3 - user defined console device (UC1:)

#### READER field (bits 2,3)

- 0 - READER is the Teletype device (TTY:)
- 1 - READER is the high-speed reader device (RDR:)
- 2 - user defined reader # 1 (UR1:)
- 3 - user defined reader # 2 (UR2:)

#### PUNCH field (bits 4,5)

- 0 - PUNCH is the Teletype device (TTY:)
- 1 - PUNCH is the high speed punch device (PUN:)
- 2 - user defined punch # 1 (UP1:)
- 3 - user defined punch # 2 (UP2:)

#### LIST field (bits 6,7)

- 0 - LIST is the Teletype device (TTY:)
- 1 - LIST is the CRT device (CRT:)
- 2 - LIST is the line printer device (LPT:)
- 3 - user defined list device (UL1:)

All Information Presented Here is Proprietary to Digital Research

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by you may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other transient program. Alternately, the PUNCH and LIST subroutines can just simply return, and the READER subroutine can return with a LAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, you can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in the BIOS is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

|        | most significant |          | least significant |          |
|--------|------------------|----------|-------------------|----------|
| IOBYTE | LIST             | PUNCH    | READER            | CONSOLE  |
|        | bits 6,7         | bits 4,5 | bits 2,3          | bits 0,1 |

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below

#### CONSOLE field (bits 0,1)

- 0 - console is assigned to the console printer (TTY:)
- 1 - console is assigned to the CRT device (CRT:)
- 2 - batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
- 3 - user defined console device (UC1:)

#### READER field (bits 2,3)

- 0 - READER is the Teletype device (TTY:)
- 1 - READER is the high-speed reader device (RDR:)
- 2 - user defined reader # 1 (UR1:)
- 3 - user defined reader # 2 (UR2:)

#### PUNCH field (bits 4,5)

- 0 - PUNCH is the Teletype device (TTY:)
- 1 - PUNCH is the high speed punch device (PUN:)
- 2 - user defined punch # 1 (UP1:)
- 3 - user defined punch # 2 (UP2:)

#### LIST field (bits 6,7)

- 0 - LIST is the Teletype device (TTY:)
- 1 - LIST is the CRT device (CRT:)
- 2 - LIST is the line printer device (LPT:)
- 3 - user defined list device (UL1:)

All Information Presented Here is Proprietary to Digital Research

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M systems use the IOBYTE except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and displayed. In any case, the IOBYTE implementation should be omitted until your basic CBIOS is fully implemented and tested, then add the IOBYTE to increase your facilities.

#### 5.4. BIOS Subroutine Entry Points.

The actions which must take place upon entry to each BIOS subroutine are given below. It should be noted that disk I/O is always performed through a sequence of calls on the various disk access subroutines which setup the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) offset and segment addresses involved in the I/O operation. After all these parameters have been setup, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a call to set the DMA segment base and a call to set the DMA offset followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE subroutines should perform several retries (10 is standard) before reporting the error condition to the BDOS. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

**INIT** This subroutine is called directly by the CP/M-86 loader after the CP/M.SYS file has been read into memory. The procedure is responsible for any hardware initialization not performed by the boot-strap-loader, setting initial values for BIOS variables (including IOBYTE), printing a sign-on message, and initializing the interrupt vector to point to the BDOS offset (0b11h) and base. When this routine completes, it jumps to the CCP offset (0h). All segment registers should be initialized at this time to contain the base of the operating system.

**WBOOT** This subroutine is called whenever a program terminates by performing a BDOS function #0 call.

All Information Presented Here is Proprietary to Digital Research

Some re-initialization of the hardware or software may occur here. When this routine completes, it jumps directly to the warm start entry point of the CCP (06H).

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONST  | Sample the status of the currently assigned console device and return 0FFH in register AL if a character is ready to read, and 00H in register AL if no console characters are ready.                                                                                                                                                                                                                                                                                                    |
| CONIN  | Read the next console character into register AL, and set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.                                                                                                                                                                                                                                                                                                   |
| CONOUT | Send the character from register CL to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which have undesirable effects on the console device.                                                       |
| LIST   | Send the character from register CL to the currently assigned listing device. The character is in ASCII with zero parity.                                                                                                                                                                                                                                                                                                                                                                |
| PUNCH  | Send the character from register CL to the currently assigned punch device. The character is in ASCII with zero parity.                                                                                                                                                                                                                                                                                                                                                                  |
| READER | Read the next character from the currently assigned reader device into register AL with zero parity (high order bit must be zero), an end of file condition is reported by returning an ASCII CONTROL-Z (1AH).                                                                                                                                                                                                                                                                           |
| HOME   | Return the disk head of the currently selected disk to the track 00 position. If your controller does not have a special feature for finding track 00, you can translate the call into a call to settrk with a parameter of 0.                                                                                                                                                                                                                                                           |
| SELDSK | Select the disk drive given by register CL for further operations, where register C contains 0 for drive A, 1 for drive B, and so-forth up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in BX the base address of the selected drive's Disk Parameter Header. For standard floppy disk drives, the content of the header and associated tables does not change, and thus the program segment included in the |

All Information Presented Here is Proprietary to Digital Research

sample CBIOS performs this operation automatically. If there is an attempt to select a non-existent drive, SELDSK returns BX=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is performed. This is due to the fact that disk select operations may take place without a subsequent disk operation and thus disk access may be substantially slower using some disk controllers. On entry to SELDSK it is possible to determine whether it is the first time the specified disk has been selected. Register DL, bit 0 (least significant bit) is a zero if the drive has not been previously selected. This information is of interest in systems which read configuration information from the disk in order to set up a dynamic disk definition table.

**SETTRK** Register CX contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register CX can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.

**SETSEC** Register CX contains the translated sector number for subsequent disk accesses on the currently selected drive (see SECTAN, below). You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

**SETDMA** Register CX contains the DMA (disk memory access) offset for subsequent read or write operations. For example, if CX = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH offset from the current DMA segment base, and all subsequent write operations get their data from that address, until the next calls to SETDMA and SETDMAB occur. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128 byte area starting at the selected DMA offset and base for the memory buffer during the following read or write operations.

**READ** Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA

All Information Presented Here is Proprietary to Digital Research

offset and segment base has been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register AL:

- 0        no errors occurred
- 1        non-recoverable error condition occurred

Currently, CP/M-86 responds only to a zero or non-zero value as the return code. That is, if the value in register AL is 0 then CP/M assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing RETURN to ignore the error, or CONTROL-C to abort.

- WRITE**        Write the data from the currently selected DMA buffer to the currently selected drive, track, and sector. The data should be marked as "non deleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register AL, with error recovery attempts as described above.
- LISTST**       Return the ready status of the list device. Used by the DESPOOL program to improve console response during its operation. The value 00 is returned in AL if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer. Note that a 00 value always suffices.
- SECTTRAN**     Performs sector logical to physical sector translation in order to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems which use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTTRAN receives a logical sector number in CX, and a translate table offset in DX. The sector number is used as an index into the translate table, with the resulting physical sector number in BX. For standard systems, the tables and

All Information Presented Here is Proprietary to Digital Research

indexing code is provided in the CBIOS and need not be changed. If DX = 0000H no translation takes place, and CX is simply copied to BX before returning. Otherwise, SECTTRAN computes and returns the translated sector number in BX.

**SETDMAB** Register CX contains the segment base for subsequent DMA read or write operations. The BIOS will use the 128 byte buffer at the memory address determined by the DMA base plus DMA offset during read and write operations.

**GETSEGB** Returns the address of the Memory Region Table (MRT) in BX. The returned value is the offset of the table relative to the start of the operating system. The table defines the location and extent of physical memory which is available for transient programs. Memory areas reserved for interrupt vectors and the CP/M-86 operating system are not included in the MRT. The Memory Region Table takes the form:

|       |        |          |
|-------|--------|----------|
| 8-bit |        |          |
| ----- |        |          |
| MRT:  | R-Cnt  |          |
| ----- |        |          |
| 0:    | R-Base | R-Length |
| ----- |        |          |
| 1:    | R-Base | R-Length |
| ----- |        |          |
| . . . |        |          |
| ----- |        |          |
| n:    | R-Base | R-Length |
| ----- |        |          |
|       | 16-bit | 16-bit   |

where R-Cnt is the number of Memory Region Descriptors (equal to n+1 in the diagram above), while R-Base and R-Length give the paragraph base and length of each physically contiguous area of memory. Again, the reserved interrupt locations, normally 0-3FFH, and the CP/M operating system are not included in this map, since these are the regions allocated to transient programs. If all memory is contiguous, the R-Cnt field is 1 and n = 0, with only a single Memory Region Descriptor which defines the region. In this case, R-Base normally addresses the first paragraph boundary beyond the last BIOS address, with an R-Length field, in paragraph units, which allows access to the highest paragraph in the region.

**GETIOB** Returns the current value of the logical to physical input/output device byte (IOBYTE) in AL. This eight bit value is used to associate physical

All Information Presented Here is Proprietary to Digital Research

devices with CP/M-86's four logical devices.

SETIOB      Use the value in CL to set the value of the IOBYTE  
             stored in the BIOS

The following section describes the exact layout and construction of the disk parameter tables referenced by various subroutines in the BIOS.

All Information Presented Here is Proprietary to Digital Research

## 6. BIOS DISK DEFINITION TABLES

Similar to CP/M-80, CP/M-86 is a table-driven operating system with a separate field-configurable Basic I/O System (BIOS). By altering specific subroutines in the BIOS presented in the previous section, CP/M-86 can be customized for operation on any RAM-based 8086 or 8088 microprocessor system.

The purpose of this section is to present the organization and construction of tables within the BIOS which define the characteristics of a particular disk system used with CP/M-86. These tables can be either hand-coded or automatically generated using the GENDEF utility provided with CP/M-86. The elements of these tables are presented below.

### 6.1. The Format of the Disk Parameter Tables.

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below

| Disk Parameter Header |      |      |      |        |     |     |     |
|-----------------------|------|------|------|--------|-----|-----|-----|
| XLT                   | 0000 | 0000 | 0000 | DIRBUF | DPB | CSV | ALV |
| 16b                   | 16b  | 16b  | 16b  | 16b    | 16b | 16b | 16b |

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is given below.

|        |                                                                                                                                                                                                                                                                                                 |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| XLT    | Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e., the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables. |
| 0000   | Scratchpad values for use within the BDOS (initial value is unimportant).                                                                                                                                                                                                                       |
| DIRBUF | Address of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.                                                                                                                                                                         |
| DPB    | Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.                                                                                                                                                             |
| CSV    | Address of a scratchpad area used for software check for changed disks. This address is different for                                                                                                                                                                                           |

All Information Presented Here is Proprietary to Digital Research

each DPH.

ALV      Address of a scratchpad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given  $n$  disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive  $n-1$ . The table thus appears as

```
DPBASE
-----
00 |XLT 00| 0000 | 0000 | 0000 |DIRBUF|DBP 00|CSV 00|ALV 00|
01 |XLT 01| 0000 | 0000 | 0000 |DIRBUF|DBP 01|CSV 01|ALV 01|
-----
      (and so-forth through)
-----
n-1|XLTh-1| 0000 | 0000 | 0000 |DIRBUF|DBPn-1|CSVn-1|ALVn-1|
-----
```

where the label DPBASE defines the offset of the DPH table relative to the DS register.

A responsibility of the SELDSK subroutine, defined in the previous section, is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```
NDISKS    EQU     4    ;NUMBER OF DISK DRIVES
.....
SELDISK:            ;SELECT DISK N GIVEN BY CL
MOV       BX,0000H    ;READY FOR ERR
CMP       CL,NDISKS   ;N BEYOND MAX DISKS?
JNB       RETURN     ;RETURN IF SO
             ;0 <= N < NDISKS
MOV       CH,0       ;DOUBLE(N)
MOV       BX,CX       ;BX = N
MOV       CL,4       ;READY FOR * 16
SHL       BX,CL       ;N = N * 16
MOV       CX,OFFSET DPBASE
ADD       BX,CX       ;DPBASE + N * 16
RETURN:   RET         ;BX = .DPH(N)
```

The translation vectors (XLT 00 through XLTh-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form

All Information Presented Here is Proprietary to Digital Research

| SPT | BSH | BLM | EXM | DSM | DRM | AL0 | AL1 | CKS | OFF |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 16b | 8b  | 8b  | 8b  | 16b | 16b | 8b  | 8b  | 16b | 16b |

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field.

|     |                                                                                                                                 |
|-----|---------------------------------------------------------------------------------------------------------------------------------|
| SPT | is the total number of sectors per track                                                                                        |
| BSH | is the data allocation block shift factor, determined by the data block allocation size.                                        |
| BLM | is the block mask which is also determined by the data block allocation size.                                                   |
| EXM | is the extent mask, determined by the data block allocation size and the number of disk blocks.                                 |
| DSM | determines the total storage capacity of the disk drive                                                                         |
| DRM | determines the total number of directory entries which can be stored on this drive AL0,AL1 determine reserved directory blocks. |
| CKS | is the size of the directory check vector                                                                                       |
| OFF | is the number of reserved tracks at the beginning of the (logical) disk.                                                        |

Although these table values are produced automatically by GENDEF, it is worthwhile reviewing the derivation of each field so that the values may be cross-checked when necessary. The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that you have selected a value for BLS, the values of BSH and BLM are shown in the table below

| BLS    | BSH | BLM |
|--------|-----|-----|
| 1,024  | 3   | 7   |
| 2,048  | 4   | 15  |
| 4,096  | 5   | 31  |
| 8,192  | 6   | 63  |
| 16,384 | 7   | 127 |

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table

All Information Presented Here is Proprietary to Digital Research

| BLS    | DSM < 256 | DSM > 255 |
|--------|-----------|-----------|
| 1,024  | 0         | N/A       |
| 2,048  | 1         | 0         |
| 4,096  | 3         | 1         |
| 8,192  | 7         | 3         |
| 16,384 | 15        | 7         |

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 and AL1 can together be considered a string of 16-bits, as shown below.

| AL0 |    |    |    |    |    |    |    | AL1 |    |    |    |    |    |    |    |
|-----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
|     |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| 00  | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08  | 09 | 10 | 11 | 12 | 13 | 14 | 15 |

where position 00 corresponds to the high order bit of the byte labelled AL0, and 15 corresponds to the low order bit of the byte labelled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the following table

| BLS    | Directory Entries |
|--------|-------------------|
| 1,024  | 32 times # bits   |
| 2,048  | 64 times # bits   |
| 4,096  | 128 times # bits  |
| 8,192  | 256 times # bits  |
| 16,384 | 512 times # bits  |

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then  $CKS = (DRM+1)/4$ , where DRM is the last directory entry number. If the media is fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and can be used as a

All Information Presented Here is Proprietary to Digital Research

mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDISK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If  $CKS = (DRM+1)/4$ , then you must reserve  $(DRM+1)/4$  bytes for directory check use. If  $CKS = 0$ , then no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as  $(DSM/8)+1$ .

The BIOS shown in Appendix D demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

## 6.2. Automatic Table Generation Using GENDEF.

The GENDEF utility supplied with CP/M-86 greatly simplifies the table construction process. GENDEF reads a file

x.DEF

containing the disk definition statements, and produces an output file

x.LIB

containing assembly language statements which define the tables necessary to support a particular drive configuration. The form of the GENDEF command is:

GENDEF x parameter-list

where x has an assumed (and unspecified) filetype of DEF, and the parameter-list optionally consists of

All Information Presented Here is Proprietary to Digital Research

```

SC      Generate Disk Parameter Comments
SO      Generate DPBASE OFFSET S
SZ      Z80, 8080, 8085 Override
$COZ    (Any of the Above)

```

The C parameter causes GENDEF to produce an accompanying comment line, similar to the output from the "STAT DSK:" utility which describes the characteristics of each defined disk. Normally, the DPBASE is defined as

```
DPBASE EQU S
```

which requires a MOV CX,OFFSET DPBASE in the SELDSK subroutine shown above. For convenience, the SO parameter produces the definition

```
DPBASE EQU OFFSET S
```

allowing a MOV CX,DPBASE in SELDSK, in order to match your particular programming practices. The SZ parameter is included to override the standard 8086/8088 mode in order to generate tables acceptable for operation with Z80, 8080, and 8085 assemblers.

The disk definition contained within X.DEF is composed with the CP/M text editor, and consists of disk definition statements identical to those accepted by the DISKDEF macro supplied with CP/M-80 Version 2. A BIOS disk definition consists of the following sequence of statements:

```

DISKS      n
DISKDEF    0,...
DISKDEF    1,...
.....
DISKDEF    n-1
.....
ENDEF

```

Each statement is placed on a single line, with optional embedded comments between the keywords, numbers, and delimiters.

The DISKS statement defines the number of drives to be configured with your system, where n is an integer in the range 1 through 16. A series of DISKDEF statements then follow which define the characteristics of each logical disk, 0 through n-1, corresponding to logical drives A through P. Note that the DISKS and DISKDEF statements generate the in-line fixed data tables described in the previous section, and thus must be placed in a non-executable portion of your BIOS, typically at the end of your BIOS, before the start of uninitialized RAM.

The ENDEF (End of Diskdef) statement generates the necessary uninitialized RAM areas which are located beyond initialized RAM in your BIOS.

All Information Presented Here is Proprietary to Digital Research

The form of the DISKDEF statement is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

|     |                                              |
|-----|----------------------------------------------|
| dn  | is the logical disk number, 0 to n-1         |
| fsc | is the first physical sector number (0 or 1) |
| lsc | is the last sector number                    |
| skf | is the optional sector skew factor           |
| bls | is the data allocation block size            |
| dks | is the disk size in bls units                |
| dir | is the number of directory entries           |
| cks | is the number of "checked" directory entries |
| ofs | is the track offset to logical track 00      |
| [0] | is an optional 1.4 compatibility flag        |

The value "dn" is the drive number being defined with this DISKDEF statement. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted or equal to 0. The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read-only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks (typically double density). This parameter ensures that ensures that no directory compression takes place, which would cause

All Information Presented Here is Proprietary to Digital Research

incompatibilities with these non-standard CP/M 1.4 versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF    i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with CP/M-80 Version 1.4, and upward compatible compatible CP/M-80 Version 2 implementations, is defined using the following statements:

```
DISKS      4
DISKDEF    0,1,26,6,1024,243,64,64,2
DISKDEF    1,0
DISKDEF    2,0
DISKDEF    3,0
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS statement generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the statement. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS statement generates a table of the form:

```
DPBASE EQU S
DPE0 DW XLTO,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1 DW XLTO,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2 DW XLTO,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3 DW XLTO,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail earlier in this section. The check and allocation vector addresses are generated by the ENDEF statement for inclusion in the ram area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTTRAN receives a translation table address of DX = 0000H, and simply returns the original logical sector from CX in the BX register. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. The table shown below, for

example, is constructed when the standard skew factor  $skf = 6$  is specified in the DISKDEF statement call:

```
XLTO EQU OFFSET $
DB 1,7,13,19,25,5,11,17,23,3,9,15,21
DB 2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF statement, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF statement. For a standard four-drive system, the ENDEF statement might produce

```
1C72 = BEGDAT EQU OFFSET $
      (data areas)
1DB0 = ENDDAT EQU OFFSET $
013C = DATSI2 EQU OFFSET $-BEGDAT
```

which indicates that uninitialized RAM begins at offset 1C72H, ends at 1DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The comment included in the LIB file by the SC parameter to GENCMD will match the output from STAT. The STAT command form

STAT d:DSK:

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

```
r: 128 Byte Record Capacity
k: Kilobyte Drive Capacity
d: 32 Byte Directory Entries
c: Checked Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks
```

### 6.3. GENDEF Output.

GENDEF produces a listing of the statements included in the DEF file at the user console (CONTROL-P can be used to obtain a printed listing, if desired). Each source line is numbered, and any errors are shown below the line in error, with a "?" beneath the item which caused the condition. The source errors produced by GENCMD are listed below:

|         |                                                                                                     |
|---------|-----------------------------------------------------------------------------------------------------|
| Bad Val | More than 16 disk defined in DISKS statement.                                                       |
| Convert | Number cannot be converted, must be constant in binary, octal, decimal, or hexadecimal as in ASM86. |
| Delimit | Missing delimiter between parameters.                                                               |
| Duplic  | Duplicate definition for a disk drive.                                                              |
| Extra   | Extra parameters occur at the end of line.                                                          |
| Length  | Keyword or data item is too long.                                                                   |
| Missing | Parameter required in this position.                                                                |
| No Disk | Referenced disk not previously defined.                                                             |
| No Stmt | Statement keyword not recognized.                                                                   |
| Numeric | Number required in this position.                                                                   |
| Range   | Number in this position is out of range.                                                            |
| Too Few | Not enough parameters provided.                                                                     |
| Quote   | Missing end quote on current line.                                                                  |

Additional errors can occur when preparing input and output files:

|                           |                                                                               |
|---------------------------|-------------------------------------------------------------------------------|
| Cannot Close ".LIB" File  | LIB file close operation unsuccessful, usually due to hardware write protect. |
| ".LIB" Disk Full          | No space for LIB file.                                                        |
| No Input File Present     | Specified DEF file not found.                                                 |
| No ".LIB" Directory Space | Cannot create LIB file due to too many files on LIB disk.                     |
| Premature End-of-File     | End of DEF file encountered unexpectedly.                                     |

Given the file TWO.DEF containing the following statements

```
disks      2
diskdef 0,1,26,6,2048,256,128,128,2
diskdef 1,1,58,,2048,1024,300,0,2
endef
```

the command

```
gencmd two Sc
```

produces the console output

```
DISKDEF Table Generator, Vers 1.0
1          DISKS      2
2          DISKDEF 0,1,58,,2048,256,128,128,2
3          DISKDEF 1,1,58,,2048,1024,300,0,2
4          ENDEF
No Error(s)
```

The resulting TWO.LIB file is brought into the skeletal assembly language program shown below, using the ASM86 INCLUDE directive. The ASM86 output listing is truncated on the right, but can be easily reproduced using GENDEF and ASM86:

75

```

=          ;      16384: 128 Byte Record Capacit
=          ;      2048: Kilobyte Drive  Capacit
=          ;      300:  32 Byte Directory Entri
=          ;      0:   Checked Directory Entri
=          ;      128: Records / Extent
=          ;      16:  Records / Block
=          ;      58:  Sectors / Track
=          ;      2:   Reserved Tracks
=          ;
=      004C      dpbl      equ      offset S      ;Disk P
= 004C 3A 00      dw      58      ;Sector
= 004E 04      db      4      ;Block
= 004F 0F      db      15      ;Block
= 0050 00      db      0      ;Extnt
= 0051 FF 03      dw      1023      ;Disk S
= 0053 2B 01      dw      299      ;Direct
= 0055 F8      db      248      ;Alloc0
= 0056 00      db      0      ;Alloc1
= 0057 00 00      dw      0      ;Check
= 0059 02 00      dw      2      ;Offset
=      0000      xltl      equ      0      ;No Tra
=      0080      als1      equ      128      ;Alloca
=      0000      css1      equ      0      ;Check
=          ;      ENDEF
=          ;
=          ;      Uninitialized Scratch Memory Fo
=          ;
=      005B      begdat      equ      offset S      ;Start
= 005B      dirbuf      rs      128      ;Direct
= 00DB      alv0      rs      als0      ;Alloc
= 00FB      csv0      rs      css0      ;Check
= 011B      alv1      rs      als1      ;Alloc
= 019B      csv1      rs      css1      ;Check
=      019B      enddat      equ      offset S      ;End of
=      0140      datsiz      equ      offset S-begdat ;Size o
= 019B 00      db      0      ;Marks
=          ;      END

```

## 7. CP/M-86 BOOTSTRAP AND ADAPTATION PROCEDURES

This section describes the components of the standard CP/M-86 distribution disk, the operation of each component, and the procedures to follow in adapting CP/M-86 to non-standard hardware.

CP/M-86 is distributed on a single density IBM compatible 8" diskette using a file format which is compatible with all previous CP/M-80 operating systems. In particular, the first two tracks are reserved for operating system and bootstrap programs, while the remainder of the diskette contains directory information which leads to program and data files. CP/M-86 is distributed for operation with the Intel SBC 86/12 computer connected to floppy disks through an Intel 204 Controller. The operation of CP/M-86 on this configuration serves as a model for other 8086 and 8088 environments, and is presented below.

The principal components of the distribution system are listed below:

- The 86/12 Bootstrap ROM (BOOT ROM)
- The Cold Start Loader (LOADER)
- The CP/M-86 System (CPM.SYS)

When installed in the SBC 86/12, the BOOT ROM becomes a part of the memory address space, beginning at byte location 0FF000H, and receives control when the system reset button is depressed. In a non-standard environment, the BOOT ROM is replaced by an equivalent initial loader and, therefore, the ROM itself is not included with CP/M-86. The BOOT ROM can be obtained from Digital Research or, alternatively, it can be programmed from the listing given in Appendix B or directly from the source file is included on the distribution disk as BOOT.A86. The responsibility of the BOOT ROM is to read the LOADER from the first two system tracks into memory and pass program control to the LOADER for execution.

### 7.1. The Cold Start Load Operation.

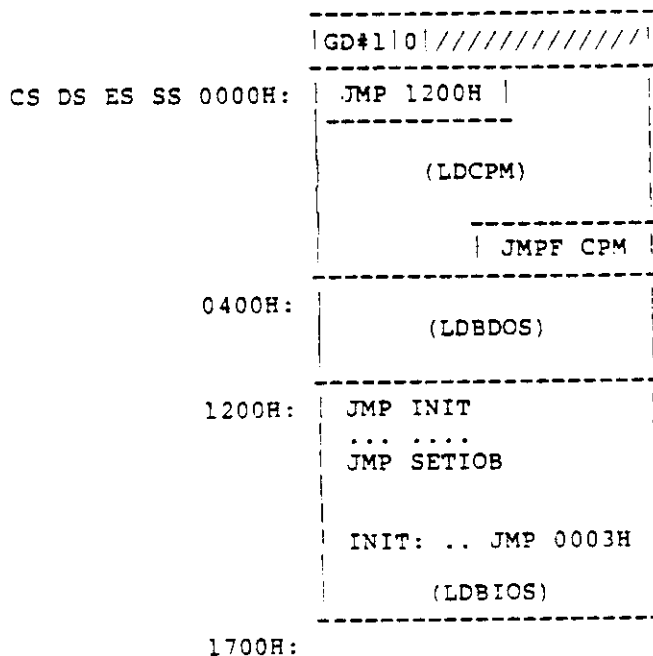
The LOADER program is a simple version of CP/M-86 that contains sufficient file processing capability to read CPM.SYS from the system disk to memory. When LOADER completes its operation, the CPM.SYS program receives control and proceeds to process operator input commands.

Both the LOADER and CPM.SYS programs are preceded by the standard CMD header record. The 128-byte LOADER header record contains the single group descriptor shown below

All Information Presented Here is Proprietary to Digital Research

| G-Form | G-Length   | A-Base | G-Min   | G-Max   |
|--------|------------|--------|---------|---------|
| 1      | xxxxxxxxxx | 0400   | xxxxxxx | xxxxxxx |
| 8b     | 16b        | 16b    | 16b     | 16b     |

where G-Form = 1 denotes a code group, "x" fields are ignored, and A-Base defines the paragraph address where the BOOT ROM begins filling memory (A-Base is the word value which is offset three bytes from the beginning of the header). Note that since only a code group is present, an 8080 memory model is assumed. Further, although the A-Base defines the base paragraph address for LOADER (byte address 04000H), the LOADER can, in fact be loaded and executed at any paragraph boundary that does not overlap CP/M-86 or the BOOT ROM. The LOADER itself consists of three parts: the Load CPM program (LDCPM), the Loader Basic Disk System (LDBDOS), and the Loader Basic I/O System (LDBIOS). Although the LOADER is setup to initialize CP/M-86 using the Intel 86/12 configuration, the LDBIOS can be field-altered to account for non-standard hardware using the same entry points described in a previous section for BIOS modification. The organization of LOADER is shown below:



where the byte offsets from the base registers are shown at the left of the diagram. GD#1 is the Group Descriptor for the LOADER code group described above, followed immediately by a "0" group terminator. The entire LOADER program is read by the BOOT ROM, excluding the header record, starting at byte location 04000H as given by the A-Field. Upon completion of the read, the BOOT ROM

All Information Presented Here is Proprietary to Digital Research

passes control to location 04000H where the LOADER program commences execution. The JMP 1200H instruction at the base of LDCPM transfers control to the beginning of the LDBIOS where control then transfers to the INIT subroutine. The subroutine starting at INIT performs device initialization, prints a sign-on message, and transfers back to the LDCPM program at byte offset 0003H. The LDCPM module opens the CPM.SYS file, loads the CP/M-86 system into memory and transfers control to CPM through the JMPF CPM instruction at the end of LDCPM execution, thus completing the cold start sequence.

The files LDCPM.H86 and LDBDOS.H86 are included with CP/M-86 so that you can append your own modified LDBIOS in the construction of a customized loader. In fact, BIOS.A86 contains a conditional assembly switch, called "loader\_bios," which, when enabled, produces the distributed LDBIOS. The INIT subroutine portion of LDBIOS is listed in Appendix C for reference purposes. To construct a custom LDBIOS, modify your standard BIOS to start the code at offset 1200H, and change your initialization subroutine beginning at INIT to perform disk and device initialization. Include a JMP to offset 0003H at the end of your INIT subroutine. Use ASM86 to assemble your LDBIOS.A86 program:

#### ASM86 LDBIOS

to produce the LDBIOS.H86 machine code file. Concatenate the three LOADER modules using PIP:

```
PIP LOADER.H86 = LDCPM.H86,LDBIOS.H86,LDBIOS.H86
```

to produce the machine code file for the LOADER program. Although the standard LOADER program ends at offset 1700H, your modified LDBIOS may differ from this last address with the restriction that the LOADER must fit within the first two tracks and not overlap CP/M-86 areas. Generate the command (CMD) file for LOADER using the GENCMD utility:

```
GENCMD LOADER 8080 CODE[A400]
```

resulting in the file LOADER.CMD with a header record defining the 8080 Memory Model with an absolute paragraph address of 400H, or byte address 4000H. The LOADER.CMD is copied to the first two tracks of a (scratch) disk under CP/M-80 using the command

```
SYSGEN LOADER.CMD
```

Alternatively, if you have access to an operational CP/M-86 system, the command

#### LDCOPY LOADER

Copies LOADER to the system tracks. You now have a diskette with a LOADER program which incorporates your custom LDBIOS capable of reading the CPM.SYS file into memory. For standardization, we assume LOADER executes at location 4000H. LOADER is statically

All Information Presented Here is Proprietary to Digital Research

relocatable, however, and its operating address is determined only by the value of A-Base in the header record.

You must, of course, perform the same function as the BOOT ROM to get LOADER into memory. The boot operation is usually accomplished in one of two ways. First, you can program your own ROM (or PROM) to perform a function similar to the BOOT ROM when your computer's reset button is pushed. As an alternative, most controllers provide a power-on "boot" operation that reads the first disk sector into memory. This one-sector program, in turn, reads the LOADER from the remaining sectors and transfers to LOADER upon completion, thereby performing the same actions as the BOOT ROM. Either of these alternatives is hardware-specific, so you'll need to be familiar with the operating environment.

## 7.2. Organization of CPM.SYS.

The CPM.SYS file, read by the LOADER program, consists of the CCP, BDOS, and BIOS in CMD file format, with a 128-byte header record similar to the LOADER program:

| G-Form | G-Length | A-Base | G-Min   | G-Max   |
|--------|----------|--------|---------|---------|
| 1      | xxxxxxxx | 040    | xxxxxxx | xxxxxxx |
| 8b     | 16b      | 16b    | 16b     | 16b     |

where, instead, the A-Base load address is paragraph 040H, or byte address 0400H immediately following the 8086 interrupt locations. The entire CPM.SYS file appears on disk as shown below.

|                    |  |  |  |  |                    |
|--------------------|--|--|--|--|--------------------|
| CS DS ES SS 0000H: |  |  |  |  | GD#1 0 //////////  |
|                    |  |  |  |  | (CCP and BDOS)     |
| 2500H:             |  |  |  |  | JMP INIT           |
|                    |  |  |  |  | ... ..             |
|                    |  |  |  |  | JMP SETIOB         |
|                    |  |  |  |  | (BIOS)             |
| 2A00H:             |  |  |  |  | INIT: .. JMP 0000H |

All Information Presented Here is Proprietary to Digital Research

where GD#1 is the Group Descriptor containing the A-Base value followed by a "0" terminator. The distributed 86/12 BIOS is listed in Appendix D, with an "include" statement that reads the SINGLES.LIB file containing the disk definition tables. The SINGLES.LIB file is created by GENDEF using the SINGLES.DEF statements shown below:

```
disks 2
diskdef 0,1,26,6,1024,243,64,64,2
diskdef 1,0
endef
```

The CPM.SYS file is read by the LOADER program beginning at the address given by A-Base (byte address 0400H), and control is passed to the INIT entry point at offset address 2500H. Any additional initialization, not performed by LOADER, takes place in the INIT subroutine and, upon completion, INIT executes a JMP 0000H to begin execution of the CCP. The actual load address of CPM.SYS is determined entirely by the address given in the A-Base field which can be changed if you wish to execute CP/M-86 in another region of memory.

Similar to the LOADER program, you can modify the BIOS by altering either the BIOS.A86 or skeletal CBIOS.A86 assembly language files which accompany your source disk. In either case, create a customized BIOS which includes your specialized I/O drivers, and assemble using ASM86:

#### ASM86 BIOS

to produce the file BIOS.H86 containing your BIOS machine code. Concatenate this new BIOS to the CPM.H86 file on your distribution disk:

```
PIP CPMX.H86 = CPM.H86, BIOS.H86
```

The resulting CCP, BDOS, and BIOS hex file is then converted to CMD file format by executing

```
GENCMD CPMX 8080 CODE[A40]
```

in order to produce the CMD memory image with A-Base = 40H. Finally, rename the CPMX file using the command

```
REN CPM.SYS = CPMX.CMD
```

and place this file on your 8086 system disk. Now the tailoring process is complete: you have replaced the BOOT ROM by either your own customized BOOT ROM, or a one-sector cold start loader which brings the LOADER program, with your custom LDBIOS, into memory at byte location 04000H. The LOADER program, in turn, reads the CPM.SYS file, with your custom BIOS, into memory at byte location 0400H. Control transfers to CP/M-86, and you are up and operating. CP/M-86 remains in memory until the next cold start operation takes

All Information Presented Here is Proprietary to Digital Research

place.

It should be noted that the two-step boot operation can be avoided if you construct a non-standard disk drive with sufficient space to hold the entire CPM.SYS file on the system tracks. In this case, the cold start brings the CP/M-86 memory image into memory at the location given by A-Base, and control transfers to the INIT entry point at offset 2500H. Thus, intermediate LOADER program is eliminated entirely, although the initialization found in the LDBIOS must, of course, take place instead within the BIOS.

Since ASM86 and GENCMD are provided in both COM and CMD formats, either CP/M-80 or CP/M-86 can be used to aid the customizing process. If CP/M-80 or CP/M-86 is not available, but you have minimal editing and debugging tools, you can write specialized disk I/O routines to read and write the system tracks, as well as the CPM.SYS file.

The two system tracks are simple to access, but the CPM.SYS file is somewhat more difficult to read. CPM.SYS is the first file on the disk and thus it appears immediately following the directory on the diskette. The directory begins on the third track, and occupies the first sixteen logical sectors of the diskette, while the CPM.SYS is found starting at the seventeenth sector. Sectors are "skewed" by a factor of six beginning with the directory track (the system tracks are sequential), so that you must load every sixth sector in reading the CPM.SYS file. Clearly, it is worth the time and effort to use an existing CP/M system to aid the conversion process.

All Information Presented Here is Proprietary to Digital Research

## 8. SECTOR BLOCKING AND DEBLOCKING.

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information which allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm which can be included within your BIOS which uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register CL:

|   |   |                                                  |
|---|---|--------------------------------------------------|
| 0 | = | normal sector write                              |
| 1 | = | write to directory sector                        |
| 2 | = | write to the first sector<br>of a new data block |

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128 byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

Appendix F lists the blocking and deblocking algorithm in skeletal form (this file is included on your CP/M disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 24 of Appendix F define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The SELDSK entry point clears the host buffer flag whenever a new disk is logged-in. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTTRAN performs a trivial trivial function of returning the physical sector number.

The principal entry points are READ and WRITE. These subroutines take the place of your previous READ and WRITE operations.

All Information Presented Here is Proprietary to Digital Research

The actual physical read or write takes place at either `WRITEHST` or `READHST`, where all values have been prepared: `hstdsk` is the host disk number, `hsttrk` is the host track number, and `hstsec` is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at `hstbuf` of length `hstsiz`. All other mapping functions are performed by the algorithms.

## 9. A SAMPLE RANDOM ACCESS PROGRAM

This manual is concluded with a rather extensive, but complete example of random access operation. The program listed in Appendix A performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.CMD, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. The only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at offset 005CH and the default buffer at offset 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For

All Information Presented Here is Proprietary to Digital Research

example, the command

```
GETKEY NAMES.DAT  LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and enhance it a bit so that it reads a sorted key file into memory. The command line might appear as:

```
QUERY NAMES.DAT LASTNAME.KEY
```

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in  $\log_2(n)$  steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well.

All Information Presented Here is Proprietary to Digital Research

# APPENDIX A: RANDOM ACCESS SAMPLE PROGRAM

```

1: ;
2: ;*****
3: ;*
4: ;*   Sample Random Access Program for CP/M-86
5: ;*
6: ;*****
7: ;
8: ;   BDOS Functions
9: ;
10: coninp equ 1 ;console input function
11: conout equ 2 ;console output function
12: pstring equ 9 ;print string until 'S'
13: rstring equ 10 ;read console buffer
14: version equ 12 ;return version number
15: openf equ 15 ;file open function
16: closef equ 16 ;close function
17: makef equ 22 ;make file function
18: readr equ 33 ;read random
19: writr equ 34 ;write random
20: ;
21: ; Equates for non graphic characters
22: cr equ 0dh ;carriage return
23: lf equ 0ah ;line feed
24: ;
25: ;
26: ; load SP, ready file for random access
27: ;
28: cseg
29: pushf ;push flags in CCP stack
30: pop ax ;save flags in AX
31: cli ;disable interrupts
32: mov bx,ds ;set SS register to base
33: mov ss,bx ;set SS, SP with interrupt
34: mov sp,offset stack ; for 80888
35: push ax ;restore the flags
36: popf
37: ;
38: ; CP/M-86 initial release returns the file
39: ; system version number of 2.2: check is
40: ; shown below for illustration purposes.
41: ;
42: mov cl,version
43: call bdos
44: cmp al,20h ;version 2.0 or later?
45: jnb versok
46: ; bad version, message and go back
47: mov dx,offset badver
48: call print
49: jmp abort
50: ;
51: versok:

```

All Information Presented Here is Proprietary to Digital Research

```

52: ;      correct version for random access
53:      mov     cl,openf      ;open default fct
54:      mov     dx,offset fcb
55:      call    bdos
56:      inc     al            ;err 255 becomes zero
57:      jnz     ready
58: ;
59: ;      cannot open file, so create it
60:      mov     cl,makef
61:      mov     dx,offset fcb
62:      call    bdos
63:      inc     al            ;err 255 becomes zero
64:      jnz     ready
65: ;
66: ;      cannot create file, directory full
67:      mov     dx,offset nospace
68:      call    print
69:      jmp     abort        ;back to ccp
70: ;
71: ;      loop back to "ready" after each command
72: ;
73: ready:
74: ;      file is ready for processing
75: ;
76:      call    readcom      ;read next command
77:      mov     ranrec,dx    ;store input record#
78:      mov     ranovf,0h    ;clear high byte if set
79:      cmp     al,'Q'       ;quit?
80:      jnz     notq
81: ;
82: ;      quit processing, close file
83:      mov     cl,closef
84:      mov     dx,offset fcb
85:      call    bdos
86:      inc     al            ;err 255 becomes 0
87:      jz      error       ;error message, retry
88:      jmps    abort       ;back to ccp
89: ;
90: ;
91: ;      end of quit command, process write
92: ;
93: ;
94: notq:
95: ;      not the quit command, random write?
96:      cmp     al,'W'
97:      jnz     notw
98: ;
99: ;      this is a random write, fill buffer until cr
100:      mov     dx,offset datmsg
101:      call    print        ;data prompt
102:      mov     cx,127       ;up to 127 characters
103:      mov     bx,offset buff ;destination
104: rloop: ;read next character to buff
105:      push    cx           ;save loop control
106:      push    bx           ;next destination

```

All Information Presented Here is Proprietary to Digital Research

```

107:      call    getchr          ;character to AL
108:      pop     bx              ;restore destination
109:      pop     cx              ;restore counter
110:      cmp     al,cr           ;end of line?
111:      jz      erloop
112: ;      not end, store character
113:      mov     byte ptr [bx],al
114:      inc     bx              ;next to fill
115:      loop    rloop           ;decrement cx ..loop if
116: erloop:
117: ;      end of read loop, store 00
118:      mov     byte ptr [bx],0h
119: ;
120: ;      write the record to selected record number
121:      mov     cl,writer
122:      mov     dx,offset fcb
123:      call    bdos
124:      or      al,al           ;error code zero?
125:      jz      ready          ;for another record
126:      jmps    error          ;message if not
127: ;
128: ;
129: ;
130: ;      end of write command, process read
131: ;
132: ;
133: notw:
134: ;      not a write command, read record?
135:      cmp     al,'R'
136:      jz      ranread
137:      jmps    error          ;skip if not
138: ;
139: ;      read random record
140: ranread:
141:      mov     cl,readr
142:      mov     dx,offset fcb
143:      call    bdos
144:      or      al,al           ;return code 00?
145:      jz      readok
146:      jmps    error
147: ;
148: ;      read was successful, write to console
149: readok:
150:      call    crlf            ;new line
151:      mov     cx,128          ;max 128 characters
152:      mov     si,offset buff  ;next to get
153: wloop:
154:      lods    al              ;next character
155:      and     al,07fh         ;mask parity
156:      jnz     wloop1
157:      jmp     ready          ;for another command if
158: wloop1:
159:      push    cx              ;save counter
160:      push    si              ;save next to get
161:      cmp     al,' '          ;graphic?

```

All Information Presented Here is Proprietary to Digital Research

```

162:      jb      skipw      ;skip output if not grab
163:      call    putchr      ;output character
164: skipw:
165:      pop      si
166:      pop      cx
167:      loop     wloop      ;decrement CX and check
168:      jmp      ready
169: ;
170: ;
171: ; end of read command, all errors end-up here
172: ;
173: ;
174: error:
175:      mov      dx,offset errmsq
176:      call     print
177:      jmp      ready
178: ;
179: ; BDOS entry subroutine
180: bdos:
181:      int      224          ;entry to BDOS if by INT
182:      ret
183: ;
184: abort:
185:      mov      cl,0          ;return to CCP
186:      call     bdos          ;use function 0 to end e
187: ;
188: ; utility subroutines for console i/o
189: ;
190: getchr:
191:      ;read next console character to a
192:      mov      cl,coninp
193:      call     bdos
194:      ret
195: ;
196: putchr:
197:      ;write character from a to console
198:      mov      cl,conout
199:      mov      dl,al          ;character to send
200:      call     bdos          ;send character
201:      ret
202: ;
203: crlf:
204:      ;send carriage return line feed
205:      mov      al,cr          ;carriage return
206:      call     putchr
207:      mov      al,lf          ;line feed
208:      call     putchr
209:      ret
210: ;
211: print:
212:      ;print the buffer addressed by dx until s
213:      push     dx
214:      call     crlf
215:      pop      dx            ;new line
216:      mov      cl,pstring

```

All Information Presented Here is Proprietary to Digital Research

```

217:      call    bdos          ;print the string
218:      ret
219: ;
220: readcom:
221:      ;read the next command line to the conbuf
222:      mov     dx,offset prompt
223:      call    print          ;command?
224:      mov     cl,rstring
225:      mov     dx,offset conbuf
226:      call    bdos          ;read command line
227: ;      command line is present, scan it
228:      mov     ax,0           ;start with 0000
229:      mov     bx,offset conlin
230: readc:  mov     dl,[bx]      ;next command character
231:      inc     bx             ;to next command positio
232:      mov     dh,0           ;zero high byte for add
233:      or      dl,dl          ;check for end of comman
234:      jnz     getnum
235:      ret
236: ;      not zero, numeric?
237: getnum:
238:      sub     dl,'0'
239:      cmp     dl,10          ;carry if numeric
240:      jnb     endrd
241:      mov     cl,10
242:      mul     cl             ;multiply accumulator by
243:      add     ax,dx          ;+digit
244:      jmps    readc          ;for another char
245: endrd:
246: ;      end of read, restore value in a and return value
247:      mov     dx,ax          ;return value in DX
248:      mov     al,-1[bx]
249:      cmp     al,'a'         ;check for lower case
250:      jnb     transl
251:      ret
252: transl: and     al,5fH      ;translate to upper case
253:      ret
254: ;
255: ;
256: ; Template for Page 0 of Data Group
257: ; Contains default FCB and DMA buffer
258: ;
259:      dseg
260:      org     05ch
261: fcb    rb      33          ;default file control bl
262: ranrec rw      1          ;random record position
263: ranovf rb      1          ;high order (overflow) b
264: buff   rb     128         ;default DMA buffer
265: ;
266: ; string data area for console messages
267: badver db      'sorry, you need cp/m version 2S'
268: nospace db      'no directory spaces'
269: datmsg  db      'type data: S'
270: errmsg  db      'error, try again.S'
271: prompt  db      'next command? S'

```

All Information Presented Here is Proprietary to Digital Research

```
272: ;
273: ;
274: ;    fixed and variable data area
275: ;
276: conbuf db      conlen ;length of console buffer
277: consiz rs      1      ;resulting size after read
278: conlin rs      32      ;length 32 buffer
279: conlen equ     offset S - offset consiz
280: ;
281:          rs      31      ;16 level stack
282: stack   rb      1
283:          db      0      ;end byte for GENCMD
284:          end
```

## APPENDIX B: LISTING OF THE BOOT ROM

```
*****
*
* This is the original BOOT ROM distributed with CP/M
* for the SBC 86/12 and 204 Controller. The listing
* is truncated on the right, but can be reproduced by
* assembling ROM.A86 from the distribution disk. Note
* that the distributed source file should always be
* referenced for the latest version
*
*****
```

```
; ROM bootstrap for CP/M-86 on an iSBC86/12
; with the
; Intel SBC 204 Floppy Disk Controller
;
; Copyright (C) 1980,1981
; Digital Research, Inc.
; Box 579, Pacific Grove
; California, 93950
;
```

```
*****
; This is the BOOT ROM which is initiated
; by a system reset. First, the ROM moves
; a copy of its data area to RAM at loca-
; tion 00000H, then initializes the segment
; registers and the stack pointer. The
; various peripheral interface chips on the
; SBC 86/12 are initialized. The 8251
; serial interface is configured for a 9600
; baud asynchronous terminal, and the in-
; terrupt controller is setup for inter-
; rupts 10H-17H (vectors at 00040H-0005FH)
; and edge-triggered auto-EOI (end of in-
; terrupt) mode with all interrupt levels
; masked-off. Next, the SBC 204 Diskette
; controller is initialized, and track 1
; sector 1 is read to determine the target
; paragraph address for LOADER. Finally,
; the LOADER on track 0 sectors 2-26 and
; track 1 sectors 1-26 is read into the
; target address. Control then transfers
; to LOADER. This program resides in two
; 2716 EPROM's (2K each) at location
; 0FF000H on the SBC 86/12 CPU board. ROM
; 0 contains the even memory locations, and
; ROM 1 contains the odd addresses. BOOT
; ROM uses RAM between 00000H and 000FFH
```

All Information Presented Here is Proprietary to Digital Research

```

; * (absolute) for a scratch area, along with*
; * the sector 1 buffer. *
; *****
00FF      true      equ      0ffh
FF00      false     equ      not true
;
00FF      debug     equ      true
; debug = true indicates bootstrap is in same rom
; with SBC 957 "Execution Vehicle" monitor
; at FE00:0 instead of FF00:0
;
000D      cr        equ      13
000A      lf        equ      10
;
;          disk ports and commands
;
00A0      base204    equ      0a0h
00A0      fdccom     equ      base204+0
00A0      fdcstat    equ      base204+0
00A1      fdcparm    equ      base204+1
00A1      fdcrslt    equ      base204+1
00A2      fdcrst     equ      base204+2
00A4      dmacadr     equ      base204+4
00A5      dmaccont    equ      base204+5
00A6      dmacscan    equ      base204+6
00A7      dmacsadr    equ      base204+7
00A8      dmacmode    equ      base204+8
00A8      dmacstat    equ      base204+8
00A9      fdcsel     equ      base204+9
00AA      fdcsegment  equ      base204+10
00AF      reset204    equ      base204+15
;
; actual console baud rate
2580      baud_rate  equ      9600
; value for 8253 baud counter
0008      baud       equ      768/(baud_rate/100)
;
00DA      csts       equ      0DAh      ; 8251 status port
00D8      cdata      equ      0D8h      ; " data port"
;
00D0      tch0       equ      0D0h      ; 8253 PIC channel 0
00D2      tch1       equ      tch0+2    ; ch 1 port
00D4      tch2       equ      tch0+4    ; ch 2 port
00D6      tcmd       equ      tch0+6    ; 8253 command port
;
00C0      icp1       equ      0C0h      ; 8259a port 0
00C2      icp2       equ      0C2h      ; 8259a port 1
;
;
;          IF NOT DEBUG
ROMSEG     EQU      0FF00H      ; normal
;          ENDIF
;
;          IF DEBUG
ROMSEG     EQU      0FE00H      ; share prom with
FE00

```

All Information Presented Here is Proprietary to Digital Research

```

                                ENDIF
                                ;
                                ;
                                ; This long jump prom'd in by hand
                                ; cseg      0ffffh      ;reset goes to
                                ; JMPF      BOTTOM      ;boot is at bo
                                ; EA 00 00 00 FF      ;cs = bottom c
                                ;                                     ip =
                                ;
                                ; EVEN PROM          ODD PROM
                                ; 7F8 - EA          7F8 - 00
                                ; 7F9 - 00          7F9 - 00
                                ; 7FA - FF          ;this is not d
                                ;
                                cseg      romseg
                                ;
                                ;First, move our data area into RAM at 0000:02
                                ;
                                0000 8C C8      mov ax,cs
                                0002 8E D8      mov ds,ax      ;point DS to CS for so
                                0004 BE 3F 01      mov SI,drombegin      ;start of data
                                0007 BF 00 02      mov DI,offset ram_start ;offset of des
                                000A B8 00 00      mov ax,0
                                000D 8E C0      mov es,ax      ;destination segment
                                000F B9 E6 00      mov CX,data_length      ;how much to
                                0012 F3 A4      rep movs al,al      ;move out of e
                                ;
                                0014 B8 00 00      mov ax,0
                                0017 8E D8      mov ds,ax      ;data segment now in R
                                0019 8E D0      mov ss,ax
                                001B BC 2A 03      mov sp,stack_offset      ;Initialize st
                                001E FC      cld      ;clear the di
                                ;
                                IF NOT DEBUG
                                ;
                                ;Now, initialize the console USART and baud rat
                                ;
                                mov al,0Eh
                                out csts,al      ;give 8251 dummy mode
                                mov al,40h
                                out csts,al      ;reset 8251 to accept
                                mov al,4Eh
                                out csts,al      ;normal 8 bit asynch m
                                mov al,37h
                                out csts,al      ;enable Tx & Rx
                                mov al,0B6h
                                out tcmd,al      ;8253 ch.2 square wave
                                mov ax,baud
                                out tch2,al      ;low of the baud rate
                                mov al,ah
                                out tch2,al      ;high of the baud rate
                                ;
                                ENDIF
                                ;
                                ;Setup the 8259 Programmable Interrupt Control
                                ;

```

All Information Presented Here is Proprietary to Digital Research

```

001F B0 13      mov al,13h
0021 E6 C0      out icp1,al      ;8259a ICW 1  3086 mode
0023 B0 10      mov al,10h
0025 E6 C2      out icp2,al      ;8259a ICW 2  vector 0
0027 B0 1F      mov al,1Fh
0029 E6 C2      out icp2,al      ;8259a ICW 4  auto EOI
002B B0 FF      mov al,0FFh
002D E6 C2      out icp2,al      ;8259a OCW 1  mask all

;
;Reset and initialize the iSBC 204 Diskette Int
;
restart:        ;also come back here on fatal e
002F E6 AF      out reset204,AL ;reset iSBC 204 logic a
0031 B0 01      mov AL,1
0033 E6 A2      out fdcrst,AL    ;give 3271 FDC
0035 B0 00      mov al,0
0037 E6 A2      out fdcrst,AL    ; a reset command
0039 BB 15 02    mov BX,offset specs1
003C E8 E1 00    CALL sendcom ;program
003F BB 18 02    mov BX,offset specs2
0042 E8 DB 00    CALL sendcom ; Shugart SA-800 drive
0045 BB 21 02    mov BX,offset specs3
0048 E8 D5 00    call sendcom ; characteristics
004B BB 10 02    homer: mov BX,offset home
004E E8 58 00    CALL execute ;home drive 0

;
0051 BB 2A 03    mov bx,sector1 ;offset for first secto
0054 B8 00 00    mov ax,0
0057 8E C0      mov es,ax        ;segment " " "
0059 E8 A7 00    call setup_dma

;
005C BB 02 02    mov bx,offset read0
005F E8 47 00    call execute ;get T0 S1

;
0062 8E 06 2D 03 mov es,ABS
0066 BB 00 00    mov bx,0 ;get loader load address
0069 E8 97 00    call setup_dma ;setup DMA to read load

;
006C BB 06 02    mov bx,offset read1
006F E8 37 00    call execute ;read track 0
0072 BB 0B 02    mov bx,offset read2
0075 E8 31 00    call execute ;read track 1

;
0078 8C 06 E8 02 mov leap_segment,ES
;
007C C7 06 E6 02 00 00 setup far jump vector
;
;
; enter LOADER
0082 FF 2E E6 02 jmpf dword ptr leap_offset

;
pmsg:
0086 8A 0F      mov cl,[BX]
0088 84 C9      test cl,cl
008A 74 76      jz return
008C E8 04 00    call conout

```

All Information Presented Here is Proprietary to Digital Research

```

008F 43          inc BX
0090 E9 F3 FF    jmp pmsg
;
conout:
0093 E4 DA      in al,csts
0095 A8 01      test al,1
0097 74 FA      jz conout
0099 8A C1      mov al,cl
009B E6 D8      out cdata,al
009D C3        ret
;
conin:
009E E4 DA      in al,csts
00A0 A8 02      test al,2
00A2 74 FA      jz conin
00A4 E4 D8      in al,cdata
00A6 24 7F      and al,7Fh
00A8 C3        ret
;
;
;
execute:        ;execute command string @ [BX]
                ;<BX> points to length,
                ;followed by Command byte
                ;followed by length-1 parameter
;
00A9 89 1E 00 02    mov     lastcom,BX      ;remember what
retry:          ;retry if not r
                call    sendcom      ;execute the com
                ;now, let's see
                ;of status poll
                ;for that comman
                ;point to comman
00B0 8B 1E 00 02    mov     BX,lastcom
00B4 8A 47 01      mov     AL,1[BX]      ;get command op
00B7 24 3F        and     AL,3fh      ;drop drive code
00B9 99 00 08      mov     CX,0800h    ;mask if it will
00BC 3C 2C        cmp     AL,2ch      ;see if interrup
00BE 72 0B        jb      execpoll
00C0 B9 80 80      mov     CX,8080h    ;else we use "no
00C3 24 0F        and     AL,0fh      ;unless . . .
00C5 3C 0C        cmp     AL,0ch      ;there isn't
00C7 B0 00        mov     AL,0
00C9 77 37        ja      return      ;any result at
;
execpoll:        ;poll for bit in b, toggled with
                in     AL,FDCSTAT
                and     AL,CH
                xor     AL,CL ! JZ execpoll
;
                in     AL,fdcrlt      ;get result req
                and     AL,leh      ;look only at r
                jz      return      ;zero means it's
;
                cmp     al,10h
                jne     fatal        ;if other than
00D9 3C 10
00DB 75 13

```

All Information Presented Here is Proprietary to Digital Research

```

00DD BB 13 02      ;      mov bx,offset rdstat
00E0 E8 3D 00      call sendcom      ;perform read s

rd_poll:          in al,fdc_stat
00E3 E4 A0          test al,80h      ;wait for commar
00E5 A8 80          jnz rd_poll
00E7 75 FA          mov bx,last_com  ;recover last a
00E9 8B 1E 00 02    jmp retry      ;and try it ove
00ED E9 BD FF      ;

fatal:            ; fatal error
00F0 B4 00          mov ah,0
00F2 8B D8          mov bx,ax      ;make 16 bits
00F4 8B 9F 27 02    mov bx,errtbl[BX]
;                print appropriate error message
00F8 E8 8B FF      call pmsg
00FB E8 A0 FF      call conin      ;wait for key s
00FE 58            pop ax          ;discard unused
00FF E9 2D FF      jmp restart     ;then start all

;
return:           RET      ;return from EX

;
setupdma:         mov AL,04h
0103 B0 04          out dmacmode,AL      ;enable dmac
0105 E6 A8          mov al,0
0107 B0 00          out dmaccont,AL      ;set first (dum
0109 E6 A5          mov AL,40h
010B B0 40          out dmaccont,AL      ;force read da
010D E6 A5          mov AX,ES
010F 8C C0          out fdcsegment,AL
0111 E6 AA          mov AL,AH
0113 8A C4          out fdcsegment,AL
0115 E6 AA          mov AX,BX
0117 8B C3          out dmacadr,AL
0119 E6 A4          mov AL,AH
011B 8A C4          out dmacadr,AL
011D E6 A4          RET
011F C3            ;
;
;
;
sendcom:          ;routine to send a command stri
0120 E4 A0          in AL,fdcstat
0122 24 80          and AL,80h
0124 75 FA          jnz sendcom      ;insure command not bus
0126 8A 0F          mov CL,[BX]      ;get count
0128 43            inc BX
0129 8A 07          mov al,[BX]      ;point to and fetch com
012B E6 A0          out fdccom,AL     ;send command

parmloop:         dec CL
012D FE C9          jz return      ;see if anv (more) para
012F 74 D1          inc BX          ;point to next paramete
0131 43            parmoll:

```

All Information Presented Here is Proprietary to Digital Research

```

0132 E4 A0          in AL,fdostat
0134 24 20          and AL,20h
0136 75 FA          jnz parmloop    ;loop until parm not ful
0138 8A 07          mov AL,[BX]
013A E6 A1          out fdcparm,AL    ;output next parameter
013C E9 EE FF          jmp parmloop    ;go see about another
;
;
;      Image of data to be moved to RAM
;
013F      drombegin equ offset S
;
013F 00 00      clastcom      dw      0000h    ;last command
;
0141 03      creadstring      db      3        ;length
0142 52          db      52h      ;read function c
0143 00          db      0        ;track #
0144 01          db      1        ;sector #
;
0145 04      creadtrk0      db      4
0146 53          db      53h      ;read multiple
0147 00          db      0        ;track 0
0148 02          db      2        ;sectors 2
0149 19          db      25       ;through 26
;
014A 04      creadtrk1      db      4
014B 53          db      53h      ;track 1
014C 01          db      1        ;sectors 1
014D 01          db      1        ;through 26
014E 1A          db      26
;
014F 02 69 00      chome0      db      2,69h,0
0152 01 6C          crdstat0    db      1,6ch
0154 05 35 0D      cspecs1     db      5,35h,0dh
0157 08 08 E9          db      08h,08h,0e9h
015A 05 35 10      cspecs2     db      5,35h,10h
015D FF FF FF      cspecs3     db      255,255,255
0160 05 35 18          db      5,35h,18h
0163 FF FF FF          db      255,255,255
;
0166 47 02      cerrtbl dw      offset er0
0168 47 02          dw      offset er1
016A 47 02          dw      offset er2
016C 47 02          dw      offset er3
016E 57 02          dw      offset er4
0170 65 02          dw      offset er5
0172 70 02          dw      offset er6
0174 7F 02          dw      offset er7
0176 90 02          dw      offset er8
0178 A2 02          dw      offset er9
017A B2 02          dw      offset erA
017C C5 02          dw      offset erB
017E D3 02          dw      offset erC
0180 47 02          dw      offset erD
0182 47 02          dw      offset erE

```

All Information Presented Here is Proprietary to Digital Research

```

0184 47 02                                dw      offset erF
;
0186 0D 0A 4E 75 6C 6C Cer0               db      cr,lf,'Null Error ??',0
      20 45 72 72 6F 72
      20 3F 3F 00
      0186                                Cer1     equ      cer0
      0186                                Cer2     equ      cer0
      0186                                Cer3     equ      cer0
0196 0D 0A 43 6C 6F 63 Cer4               db      cr,lf,'Clock Error',0
      6B 20 45 72 72 6F
      72 00
01A4 0D 0A 4C 61 74 65 Cer5               db      cr,lf,'Late DMA',0
      20 44 4D 41 00
01AF 0D 0A 49 44 20 43 Cer6               db      cr,lf,'ID CRC Error',0
      52 43 20 45 72 72
      6F 72 00
01BE 0D 0A 44 61 74 61 Cer7               db      cr,lf,'Data CRC Error',0
      20 43 52 43 20 45
      72 72 6F 72 00
01CF 0D 0A 44 72 69 76 Cer8               db      cr,lf,'Drive Not Ready',0
      65 20 4E 6F 74 20
      52 65 61 64 79 00
01E1 0D 0A 57 72 69 74 Cer9               db      cr,lf,'Write Protect',0
      65 20 50 72 6F 74
      65 63 74 00
01F1 0D 0A 54 72 6B 20 CerA               db      cr,lf,'Trk 00 Not Found',0
      30 30 20 4E 6F 74
      20 46 6F 75 6E 64
      00
0204 0D 0A 57 72 69 74 CerB               db      cr,lf,'Write Fault',0
      65 20 46 61 75 6C
      74 00
0212 0D 0A 53 65 63 74 CerC               db      cr,lf,'Sector Not Found',0
      6F 72 20 4E 6F 74
      20 46 6F 75 6E 64
      00
      0186                                CerD     equ      cer0
      0186                                CerE     equ      cer0
      0186                                CerF     equ      cer0
;
0225                                dromend equ offset S
;
00E6                                data_length equ dromend-drombegin
;
;                                reserve space in RAM for data area
;                                (no hex records generated here)
;
0000                                dseg      0
                                org        0200h
;
0200                                ram_start equ      S
0200                                lastcom   rw        1        ;last command
0202                                read0     rb        4        ;read track 0 s
0206                                read1     rb        5        ;read T0 S2-26
020B                                read2     rb        5        ;read T1 S1-26

```

All Information Presented Here is Proprietary to Digital Research

```

0210      home      rb      3      ;home drive 0
0213      rdstat   rb      2      ;read status
0215      specs1   rb      6
021B      specs2   rb      6
0221      specs3   rb      6
0227      errtbl   rw      16
0247      er0      rb      length cer0      ;16
      0247      er1      equ      er0
      0247      er2      equ      er0
      0247      er3      equ      er0
0257      er4      rb      length cer4      ;14
0265      er5      rb      length cer5      ;11
0270      er6      rb      length cer6      ;15
027F      er7      rb      length cer7      ;17
0290      er8      rb      length cer8      ;18
02A2      er9      rb      length cer9      ;16
02B2      erA      rb      length cerA      ;19
02C5      erB      rb      length cerB      ;14
02D3      erC      rb      length cerC      ;19
      0247      erD      equ      er0
      0247      erE      equ      er0
      0247      erF      equ      er0
;
02E6      leap_offset  rw      1
02E8      leap_segment  rw      1
;
;
02EA      ;local stack
      032A      stack_offset  equ      offset S;stack from here
;
;      T0 S1 read in here
      032A      sector1      equ      offset S
;
032A      Ty      rb      1
032B      Len      rw      1
032D      Abs      rw      1      ;ABS is all we c
032F      Min      rw      1
0331      Max      rw      1
end

```

## APPENDIX C: LDBIOS LISTING

```
*****
*
* This the the LOADER BIOS, derived from the BIOS
* program by enabling the "loader_bios" condi-
* tional assembly switch. The listing has been
* edited to remove portions which are duplicated
* in the BIOS listing which appears in Appendix D
* where ellipses "..." denote the deleted portions
* (the listing is truncated on the right, but can
* be reproduced by assembling the BIOS.A86 file
* provided with CP/M-86)
*
*****
```

```
*****
;
; * Basic Input/Output System (BIOS) for
; * CP/M-86 Configured for iSBC 86/12 with
; * the iSBC 204 Floppy Disk Controller
;
; * (Note: this file contains both embedded
; * tabs and blanks to minimize the list file
; * width for printing purposes. You may wish
; * to expand the blanks before performing
; * major editing.)
*****
```

```
; Copyright (C) 1980,1981
; Digital Research, Inc.
; Box 579, Pacific Grove
; California, 93950
```

```
; (Permission is hereby granted to use
; or abstract the following program in
; the implementation of CP/M, MP/M or
; CP/NET for the 8086 or 8088 Micro-
; processor)
```

```
FFFF      true      equ -1
0000      false     equ not true
```

```
*****
;
; * Loader_bios is true if assembling the
; * LOADER BIOS, otherwise BIOS is for the
; * CPM.SYS file. Bld_list is true if we
; * have a serial printer attached to BLC8538
; * 3dos_int is interrupt used for earlier
*****
```

All Information Presented Here is Proprietary to Digital Research

```

; * versions.
; *
; *****
loader_bios equ true
bld_list equ true
bdos_int equ 224 ;BDOS Interrupt

IF not loader_bios
;-----
; . . .
;-----
ENDIF ;not loader_bios

IF loader_bios
;-----
bios_code equ 1200h ;start of LDBIOS
ccp_offset equ 0003h ;base of CPMLOADER
bdos_ofst equ 0406h ;stripped BDOS entry
;-----
ENDIF ;loader_bios
. . .

cseg
org ccpoffset
ccp:
org bios_code

; *****
; *
; * BIOS Jump Vector for Individual Routines *
; *
; *****

1200 E9 3C 00 jmp INIT ;Enter from BOOT ROM or LOADER
1203 E9 61 00 jmp WBOOT ;Arrive here from BDOS call 0

1239 E9 64 00 jmp GETIOBF ;return I/O map byte (IOBYTE)
123C E9 64 00 jmp SETIOBF ;set I/O map byte (IOBYTE)

; *****
; *
; * INIT Entry Point, Differs for LDBIOS and *
; * BIOS, according to "Loader_Bios" value *
; *
; *****

INIT: ;print signon message and initialize ha
123F 8C C8 mov ax,cs ;we entered with a JMP
1241 8E D0 mov ss,ax ; CS: as the initial va
1243 8E D8 mov ds,ax ; DS:,

```

All Information Presented Here is Proprietary to Digital Research

## LDBIOS Listing

## Appendix C

```

1245 8E C0      mov es,ax      ; and ES:
                ; use local stack during initialization
1247 BC A9 16    mov sp,offset stkbases
124A FC         cld           ; set forward direction

                IF      not loader_bios
                -----
                ; This is a BIOS for the CPM.SYS file.
                . . .
                -----
                ENDIF    ;not loader_bios

                IF      loader_bios
                -----
                ; This is a BIOS for the LOADER
124B 1E         push ds      ; save data segment
124C B8 00 00    mov ax,0
124F 8E D8      mov ds,ax    ; point to segment zero
                ; BDOS interrupt offset
1251 C7 06 80 03 06 0B  mov bdos_offset,bdos_ofst
1257 8C 0E 16 00  mov bdos_segment,CS ; bdos interrupt seg
125B 1F         pop ds      ; restore data segment
                -----
                ENDIF    ;loader_bios

125C BB 15 14    mov bx,offset signon
125F E8 5A 00    call msg     ; print signon message
1262 B1 00      mov ci,0      ; default to dr A: on co
1264 E9 9C ED    jmp ccp      ; jump to cold start ent

WBOOT: jmp ccp+6      ; direct entry to CCP at

                IF      not loader_bios
                -----
                . . .
                -----
                ENDIF    ;not loader_bios

                *****
                *
                * CP/M Character I/O Interface Routines
                * Console is Usart (i8251a) on iSBC 86/12
                * at ports D8/DA
                *
                *****

CONST:          ; console status
                in al,csts
                . . .

126A E4 DA

```

All Information Presented Here is Proprietary to Digital Research

```

const_ret:
1272 C3      ret                ;Receiver Data Availabl

CONIN:      ;console input
1273 E8 F4 FF      call const
               . . .

CONOUT:     ;console output
127D E4 DA      in al,csts
               . . .

LISTOUT:    ;list device output
               IF      blc_list
;-----
;|
;|      call LISTST
;|      . . .
;|
;-----
               ENDIF    ;blc_list

1291 C3      ret

LISTST:     ;poll list status
               IF      blc_list
;-----
;|
;|      in al,lsts
;|      . . .
;|
;-----
               ENDIF    ;blc_list

1292 E4 41

129C C3      ret

PUNCH:     ;not implemented in this configuration
READER:
129D B0 1A      mov al,lah
129F C3      ret                ;return EOF for now

GETIOBF:
12A0 B0 00      mov al,0        ;TTY: for consistency
12A2 C3      ret                ;IOBYTE not implemented

SETIOBF:
12A3 C3      ret                ;iobyte not implemented

zero_ret:
12A4 24 00      and al,0
12A6 C3      ret                ;return zero in AL and

; Routine to get and echo a console character
; and shift it to upper case

```

All Information Presented Here is Proprietary to Digital Research

```

12A7 E8 C9 FF      uconecho:      call CONIN      ;get a console character
;*****
;*
;*          Disk Input/Output Routines
;*
;*****
12CA BB 00 00      SELDSK:          ;select disk given by register
                        mov bx,0000h
                        . . .
12EB C6 06 31 15 00 HOME:          ;move selected disk to home position (0)
                        mov trk,0      ;set disk i/o to track
                        . . .
1300 88 0E 31 15      SETTRK:      ;set track address given by CX
1304 C3              mov trk,cl      ;we only use 8 bits of
                        ret
1305 88 0E 32 15      SETSEC:      ;set sector number given by cx
1309 C3              mov sect,cl     ;we only use 8 bits of
                        ret
130A 8B D9          SECTTRAN:      ;translate sector CX using table at [D
                        mov bx,cx
                        . . .
1311 89 0E 2A 15      SETDMA:      ;set DMA offset given by CX
1315 C3              mov dma_adr,CX
                        ret
1316 89 0E 2C 15      SETDMAB:     ;set DMA segment given by CX
131A C3              mov dma_seg,CX
                        ret
;
131B BB 38 15      GETSEGT:      ;return address of physical memory ta
131E C3              mov bx,offset seq_table
                        ret
;*****
;*
;* All disk I/O parameters are setup: the *
;* Read and Write entry points transfer one *
;* sector of 128 bytes to/from the current *
;* DMA address using the current disk drive *
;*
;*****
131F B0 12          READ:          mov al,12h      ;basic read sector comm
1321 EB 02              jmps r_w_common

```

All Information Presented Here is Proprietary to Digital Research

```

WRITE:
1323 B0 0A      mov al,0ah      ;basic write sector com

r_w_common:
1325 BB 2F 15    mov bx,offset io_com ;point to command
                . . .

;*****
;*
;*          Data Areas
;*
;*
;*****
1415 data_offset equ offset $

                dseg
                org data_offset ;contiguous with
                IF loader_bios

;-----
;|
;| signon db cr,lf,cr,lf
1415 0D 0A 0D 0A  signon db 'CP/M-86 Version 2.2',cr,lf,0
1419 43 50 2F 4D 2D 38
        36 20 56 65 72 73
        69 6F 6E 20 32 2E
        32 0D 0A 00
;|
;|-----
;|          ENDIF ;loader_bios
;|
;|          IF not loader_bios
;|-----
;|          . . .
;|-----
;|          ENDIF ;not loader_bios

142F 0D 0A 48 6F 6D 65 bad_hom db cr,lf,'Home Error',cr,lf,0
=
=          include singles.lib ;read in disk definitions
=          DISKS 2
= 1541      dpbase equ $          ;Base of Disk 0
= 1668 00    db 0                ;Marks End of Disk 0

1669      loc_stk rw 32 ;local stack for initialization
16A9      stkbase equ offset $

16A9 00      db 0 ;fill last address for GENCMD

;*****
;*
;*          Dummy Data Section
;*
;*****

```

All Information Presented Here is Proprietary to Digital Research

```
0000      ;*
      ;*****
      dseg    0      ;absolute low memory
      org     0      ;(interrupt vectors)
      . . .
      END
```

# APPENDIX D: BIOS LISTING

```
*****
*
* This is the CP/M-86 BIOS, derived from the BIOS
* program by disabling the "loader_bios" condi-
* tional assembly switch. The listing has been
* truncated on the right, but can be reproduced
* by assembling the BIOS.A86 file provided with
* CP/M-86. This BIOS allows CP/M-86 operation
* with the Intel SBC 86/12 with the SBC 204 con-
* troller. Use this BIOS, or the skeletal CBIOS
* listed in Appendix E, as the basis for a cus-
* tomized implementation of CP/M-86.
* provided with CP/M-86)
*
*****
```

```
*****
;
; * Basic Input/Output System (BIOS) for
; * CP/M-86 Configured for iSBC 86/12 with
; * the iSBC 204 Floppy Disk Controller
;
; * (Note: this file contains both embedded
; * tabs and blanks to minimize the list file
; * width for printing purposes. You may wish
; * to expand the blanks before performing
; * major editing.)
*****
```

```
;
; Copyright (C) 1980,1981
; Digital Research, Inc.
; Box 579, Pacific Grove
; California, 93950
;
```

```
;
; (Permission is hereby granted to use
; or abstract the following program in
; the implementation of CP/M, MP/M or
; CP/NET for the 8086 or 8088 Micro-
; processor)
;
```

```
FFFF      true      equ -1
0000      false     equ not true
```

```
*****
;
; * Loader_bios is true if assembling the
; * LOADER_BIOS, otherwise BIOS is for the
; * CPM.SYS file. 9lc_list is true if we
*****
```

All Information Presented Here is Proprietary to Digital Research

```

; * have a serial printer attached to BLC8538 *
; * Bdos_int is interrupt used for earlier *
; * versions. *
; *
; *****
0000 loader_bios equ false
FFFF blc_list equ true
00E0 bdos_int equ 224 ;reserved BDOS Interrupt

IF not loader_bios
;-----
;|
2500 bios_code equ 2500h
0000 ccp_offset equ 0000h
0B06 bdos_ofst equ 0B06h ;BDOS entry point
;|
;-----
ENDIF ;not loader_bios

IF loader_bios
;-----
;|
bios_code equ 1200h ;start of LDBIOS
ccp_offset equ 0003h ;base of CPMLOADER
bdos_ofst equ 0406h ;stripped BDOS entry
;|
;-----
ENDIF ;loader_bios

00DA csts equ 0DAh ;i8251 status port
00D8 cdata equ 0D8h ; " data port

IF blc_list
;-----
;|
0041 lsts equ 41h ;2651 No. 0 on BLC8538
0040 ldata equ 40h ; " " " "
0060 blc_reset equ 60h ;reset selected USARTS
;|
;-----
ENDIF ;blc_list

; *****
; *
; * Intel iSBC 204 Disk Controller Ports *
; *
; *****

00A0 base204 equ 0a0h ;SBC204 assigned
00A0 fdc_com equ base204+0 ;8271 FDC out c
00A0 fdc_stat equ base204+0 ;8271 in status
00A1 fdc_parm equ base204+1 ;8271 out param
00A1 fdc_rslt equ base204+1 ;8271 in result

```

All Information Presented Here is Proprietary to Digital Research

```

00A2      fdc_rst      equ base204+2      ;8271 out reset
00A4      dmac_adr     equ base204+4      ;8257 DMA base
00A5      dmac_cont    equ base204+5      ;8257 out contr
00A6      dmac_scan    equ base204+6      ;8257 out scan
00A7      dmac_sadr    equ base204+7      ;8257 out scan
00A8      dmac_mode    equ base204+8      ;8257 out mode
00A8      dmac_stat    equ base204+8      ;8257 in status
00A9      fdc_sel      equ base204+9      ;FDC select por
00AA      fdc_segment  equ base204+10     ;segment address
00AF      reset_204    equ base204+15     ;reset entire i

```

```

000A      max_retries  equ 10              ;max retries on
                                           ;before perm er
000D      cr           equ 0dh             ;carriage retur
000A      lf           equ 0ah             ;line feed

```

```

                cseg
                org      ccpoffset
ccp:
                org      bios_code

```

```

;*****
;*
;* BIOS Jump Vector for Individual Routines *
;*
;*****

```

```

2500 E9 3C 00      jmp INIT          ;Enter from BOOT ROM or LOADER
2503 E9 84 00      jmp WBOOT        ;Arrive here from BDOS call 0
2506 E9 90 00      jmp CONST        ;return console keyboard status
2509 E9 96 00      jmp CONIN        ;return console keyboard char
250C E9 9D 00      jmp CONOUT       ;write char to console device
250F E9 A5 00      jmp LISTOUT      ;write character to list device
2512 E9 B7 00      jmp PUNCH        ;write character to punch devic
2515 E9 B4 00      jmp READER       ;return char from reader device
2518 E9 FF 00      jmp HOME         ;move to trk 00 on cur sel driv
251B E9 DB 00      jmp SELDSK       ;select disk for next rd/write
251E E9 0E 01      jmp SETTRK       ;set track for next rd/write
2521 E9 10 01      jmp SETSEC       ;set sector for next rd/write
2524 E9 19 01      jmp SETDMA       ;set offset for user buff (DMA)
2527 E9 24 01      jmp READ         ;read a 128 byte sector
252A E9 25 01      jmp WRITE        ;write a 128 byte sector
252D E9 91 00      jmp LISTST       ;return list status
2530 E9 06 01      jmp SECTTRAN     ;xlate logical->physical sector
2533 E9 0F 01      jmp SETDMAB      ;set seg base for buff (DMA)
2536 E9 11 01      jmp GETSEGT      ;return offset of Mem Desc Tabl
2539 E9 93 00      jmp GETIOBF      ;return I/O map byte (IOBYTE)
253C E9 93 00      jmp SETIOBF      ;set I/O map byte (IOBYTE)

```

```

;*****
;*
;* INIT Entry Point, Differs for LDBIOS and *
;* BIOS, according to "Loader_Bios" value *
;*
;*****

```

All Information Presented Here is Proprietary to Digital Research

```

INIT:      ;print signon message and initialize hardware
253F 8C C8      mov ax,cs      ;we entered with a JMPF
2541 8E D0      mov ss,ax      ; CS: as the initial va
2543 8E D8      mov ds,ax      ; DS:,
2545 8E C0      mov es,ax      ; and ES:
                ;use local stack during initialization
2547 BC E4 29    mov sp,offset stkbases
254A FC          cld            ;set forward direction

                IF      not loader_bios
;-----
;|
                ; This is a BIOS for the CPM.SYS file.
                ; Setup all interrupt vectors in low
                ; memory to address trap

254B 1E          push ds        ;save the DS register
254C B8 00 00    mov ax,0
254F 8E D8      mov ds,ax
2551 8E C0      mov es,ax      ;set ES and DS to zero
                ;setup interrupt 0 to address trap routine
2553 C7 06 00 00 8D 25    mov int0_offset,offset int_trap
2559 8C 0E 02 00    mov int0_segment,CS
255D BF 04 00      mov di,4
2560 BE 00 00      mov si,0      ;then propagate
2563 B9 FE 01      mov cx,510    ;trap vector to
2566 F3 A5          rep movs ax,ax ;all 256 interrupts
                ;BDOS offset to proper interrupt
2568 C7 06 80 03 06 0B    mov bdos_offset,bdos_ofst
256E 1F          pop ds        ;restore the DS register

;*****
;*
;* National "BLC 8538" Channel 0 for a serial*
;* 9600 baud printer - this board uses 8 Sig-*
;* netics 2651 Usarts which have on-chip baud*
;* rate generators.
;*
;*****

256F B0 FF          mov al,0FFh
2571 E6 60          out b/c_reset,al ;reset all usarts on
2573 B0 4E          mov al,4Eh
2575 E6 42          out ldata+2,al ;set usart 0 in async
2577 B0 3E          mov al,3Eh
2579 E6 42          out ldata+2,al ;set usart 0 to 9600 b
257B B0 37          mov al,37h
257D E6 43          out ldata+3,al ;enable Tx/Rx, and set
;|
;-----
                ENDIF      ;not loader_bios

                IF      loader_bios
;-----

```

All Information Presented Here is Proprietary to Digital Research

```

;|
;|      ;This is a BIOS for the LOADER
;|      push ds          ;save data segment
;|      mov ax,0
;|      mov ds,ax        ;point to segment zero
;|      ;BDOS interrupt offset
;|      mov bdos_offset,bdos_ofst
;|      mov bdos_segment,CS ;bdos interrupt seg
;|      pop ds          ;restore data segment
;|
;|-----|
;|      ENDIF      ;loader_bios
;|
;|      mov bx,offset signon
;|      call pmsg      ;print signon message
;|      mov cl,0        ;default to dr A: on co
;|      jmp ccp         ;jump to cold start ent
;|
257F BB 44 27
2582 E8 66 00
2585 B1 00
2587 E9 76 DA
;|
258A E9 79 DA      WBOOT: jmp ccp+6      ;direct entry to CCP at
;|
;|      IF          not loader_bios
;|-----|
;|      int_trap:
;|      cli          ;block interrupts
;|      mov ax,cs
;|      mov ds,ax    ;get our data segment
;|      mov bx,offset int_trp
;|      call pmsg
;|      hlt          ;hardstop
;|
;|-----|
;|      ENDIF      ;not loader_bios
;|
;|*****|
;|*      CP/M Character I/O Interface Routines      *|
;|*      Console is Usart (i8251a) on iSBC 86/12    *|
;|*      at ports D8/DA                              *|
;|*      *****|
;|
CONST:          ;console status
;|      in al,csts
;|      and al,2
;|      jz const_ret
;|      or al,255      ;return non-zero if RDA
;|
const_ret:
;|      ret          ;Receiver Data Availabl
;|
CONIN:          ;console input
;|      call const
;|      jz CONIN      ;wait for RDA
;|      in al,cdata
;|      and al,7fh    ;read data and remove p
;|
2599 E4 DA
259B 24 02
259D 74 02
259F 0C FF
25A1 C3
25A2 E8 F4 FF
25A5 74 FB
25A7 E4 D8
25A9 24 7F

```

All Information Presented Here is Proprietary to Digital Research

```

25AB C3                                ret

CONOUT:                                ;console output
in al,csts
and al,1                                ;get console status
jz CONOUT                                ;wait for TBE
mov al,cl
out cdata,al                            ;Transmitter Buffer Emp
ret                                      ;then return data

LISTOUT:                                ;list device output
IF blc_list
;-----
;|
25B7 E8 07 00    call LISTST
25BA 74 FB      jz LISTOUT                                ;wait for printer not b
25BC 8A C1      mov al,cl
25BE E6 40      out ldata,al                            ;send char to TI 810
;|
;-----
ENDIF ;blc_list

25C0 C3                                ret

LISTST:                                ;poll list status
IF blc_list
;-----
;|
25C1 E4 41      in al,lsts
25C3 24 81      and al,81h                                ;look at both TxRDY and
25C5 3C 81      cmp al,81h
25C7 75 0A      jnz zero_ret                                ;either false, printer
25C9 0C FF      or al,255                                    ;both true, LPT is read
;|
;-----
ENDIF ;blc_list

25CB C3                                ret

PUNCH: ;not implemented in this configuration
READER:
25CC B0 1A      mov al,lah
25CE C3                                ret                                ;return EOF for now

GETIOBF:
25CF B0 00      mov al,0                                ;TTY: for consistency
25D1 C3                                ret                                ;IOBYTE not implemented

SETIOBF:
25D2 C3                                ret                                ;iobyte not implemented

zero_ret:
25D3 24 00      and al,0

```

All Information Presented Here is Proprietary to Digital Research

```

25D5 C3          ret                ;return zero in AL and
; Routine to get and echo a console character
; and shift it to upper case

uconecho:
25D6 E8 C9 FF    call CONIN        ;get a console character
25D9 50          push ax
25DA 8A C8       mov cl,al         ;save and
25DC E8 CD FF    call CONOUT       ;echo to console
25DF 58          pop ax
25E0 3C 61       cmp al,'a'
25E2 72 06       jb uret          ;less than 'a' is ok
25E4 3C 7A       cmp al,'z'
25E6 77 02       ja uret          ;greater than 'z' is ok
25E8 2C 20       sub al,'a'-'A'   ;else shift to caps
uret:
25EA C3          ret

; utility subroutine to print messages

pmsg:
25EB 8A 07       mov al,[BX]      ;get next char from message
25ED 84 C0       test al,al
25EF 74 28       jz return        ;if zero return
25F1 8A C8       mov CL,AL
25F3 E8 B6 FF    call CONOUT      ;print it
25F6 43          inc BX
25F7 EB F2       jmps pmsg        ;next character and loop

;*****
; *
; *          Disk Input/Output Routines
; *
;*****

SELDSK:          ;select disk given by register
25F9 BB 00 00    mov bx,0000h
25FC 80 F9 02    cmp cl,2        ;this BIOS only support
25FF 73 18       jnb return       ;return w/ 0000 in BX
2601 B0 80       mov al, 80h
2603 80 F9 00    cmp cl,0
2606 75 02       jne sell         ;drive 1 if not zero
2608 B0 40       mov al, 40h      ;else drive is 0
260A A2 69 28    sell: mov sel_mask,al ;save drive select mask
;now, we need disk parameters
260D B5 00       mov ch,0
260F 8B D9       mov bx,cx        ;BX = word(CL)
2611 B1 04       mov cl,4
2613 D3 E3       shl bx,cl        ;multiply drive code *
;create offset from Disk Parameter Base
2615 81 C3 7C 28 add bx,offset dp_base
return:
2619 C3          ret

```

All Information Presented Here is Proprietary to Digital Research

```

HOME:      ;move selected disk to home position (T
261A C6 06 6C 28 00      mov trk,0          ;set disk i/o to track
261F BB 6E 28            mov bx,offset hom_com
2622 E8 35 00            call execute
2625 74 F2              jz return          ;home drive and return
2627 BB 6A 27            mov bx,offset bad_hom ;else print
262A E8 BE FF            call pmsg        ;"Home Error"
262D EB EB              jmps home         ;and retry

SETTRK:    ;set track address given by CX
262F 88 0E 6C 28        mov trk,cl        ;we only use 8 bits of
2633 C3                ret

SETSEC:    ;set sector number given by CX
2634 88 0E 6D 28        mov sect,cl       ;we only use 8 bits of
2638 C3                ret

SECTTRAN:  ;translate sector CX using table at [D
2639 8B D9              mov bx,cx
263B 03 DA              add bx,dx          ;add sector to tran tab
263D 8A 1F              mov bl,[bx]       ;get logical sector
263F C3                ret

SETDMA:    ;set DMA offset given by CX
2640 89 0E 65 28        mov dma_adr,CX
2644 C3                ret

SETDMAB:   ;set DMA segment given by CX
2645 89 0E 67 28        mov dma_seg,CX
2649 C3                ret

;
GETSEGT:   ;return address of physical memory to
264A BB 73 28          mov bx,offset seg_table
264D C3                ret

;*****
;
; * All disk I/O parameters are setup: the *
; * Read and Write entry points transfer one *
; * sector of 128 bytes to/from the current *
; * DMA address using the current disk drive *
; *
;*****

READ:      mov al,12h      ;basic read sector comm
264E B0 12
2650 EB 02      jmps r_w_common

WRITE:     mov al,0ah      ;basic write sector com
2652 B0 0A

r_w_common:
2654 BB 6A 28      mov bx,offset io_com ;point to command
2657 88 47 01      mov byte ptr 1[BX],al ;put command into
; fall into execute and return

```

All Information Presented Here is Proprietary to Digital Research

```

execute: ;execute command string.
        ;[BX] points to length,
        ; followed by Command byte,
        ; followed by length-1 parameter

265A 89 1E 63 28      mov last_com,BX ;save command address f
outer_retry:
        ;allow some retrying
265E C6 06 62 28 0A    mov rtry_cnt,max_retries
retry:
2663 8B 1E 63 28      mov BX,last_com
2667 E8 89 00          call send_com ;transmit command to i8
        ; check status poll

266A 9B 1E 63 28      mov BX,last_com
266E 8A 47 01          mov al,[bx] ;get command op code
2671 B9 00 08          mov cx,0800h ;mask if it will be "in
2674 3C 2C             cmp al,2ch
2676 72 0B             jb exec_poll ;ok if it is an interrupt
2678 B9 80 80          mov cx,8080h ;else we use "not comma
267B 24 0F             and al,0fh
267D 3C 0C             cmp al,0ch ;unless there isn't
267F B0 00             mov al,0
2681 77 36             ja exec_exit ; any result
                                ;poll for bits in CH,
                                ; toggled with bits in
exec_poll:
2683 E4 A0             in al,fdc_stat ;read status
2685 22 C5             and al,ch
2687 32 C1             xor al,cl ; isolate what we want
2689 74 F8             jz exec_poll ;and loop until it is d
                                ;Operation complete,
268B E4 A1             in al,fdc_rslt ; see if result code in
268D 24 1E             and al,1eh
268F 74 28             jz exec_exit ;no error, then exit
                                ;some type of error occ
2691 3C 10             cmp al,10h
2693 74 25             je dr_rdy ;was it a not ready dri
                                ;no,
dr_rdy: ; then we just retry read or write
2695 FE 0E 62 28      dec rtry_cnt
2699 75 C8             jnz retry ; up to 10 times

        ; retries do not recover from the
        ; hard error

269B B4 00             mov ah,0
269D 8B D8             mov bx,ax ;make error code 16 bit
269F 8B 9F 91 27      mov bx,errtbl[BX]
26A3 E8 45 FF          call pmsg ;print appropriate mess
26A6 E4 D8             in al,cdata ;flush usart receiver b
26A8 E8 2B FF          call uconecho ;read upper case consol
26AB 3C 43             cmp al,'C'

```

All Information Presented Here is Proprietary to Digital Research

```

26AD 74 25          je wboot_1      ;cancel
26AF 3C 52          cmp al,'R'
26B1 74 AB          je outer_retry ;retry 10 more times
26B3 3C 49          cmp al,'I'
26B5 74 1A          je z_ret        ;ignore error
26B7 0C FF          or al,255       ;set code for permanent
exec_exit:
26B9 C3            ret

dr_nrdy:           ;here to wait for drive ready
26BA E8 1A 00       call test_ready
26BD 75 A4          jnz retry       ;if it's ready now we a
26BF E8 15 00       call test_ready
26C2 75 9F          jnz retry       ;if not ready twice in
26C4 BB 02 28       mov bx,offset nrdymsg
26C7 E8 21 FF       call pmsg ;"Drive Not Ready"

nrdy01:            call test_ready
26CA E8 0A 00       jz nrdy01        ;now loop until drive r
26CD 74 FB          jmps retry      ;then go retry without
26CF EB 92

zret:              and al,0
26D1 24 00          ret              ;return with no error c
26D3 C3

wboot_1:           ;can't make it w/ a sho
26D4 E9 B3 FE       jmp WBOOT

;*****
;*
;* The 18271 requires a read status command *
;* to reset a drive-not-ready after the   *
;* drive becomes ready                    *
;*
;*****

test_ready:
26D7 B6 40          mov dh, 40h      ;proper mask if dr 1
26D9 F6 06 69 28 80 test sel mask,80h
26DE 75 02          jnz nrdy2
26E0 B6 04          mov dh, 04h      ;mask for dr 0 status b

nrdy2:             mov bx,offset rds_com
26E2 BB 71 28       call send_com
26E5 E8 0B 00

dr_poll:           in al,fdc_stat   ;get status word
26E8 E4 A0          test al,80h
26EA A8 80          jnz dr_poll     ;wait for not command b
26EC 75 FA          in al,fdc_rslt  ;get "special result"
26EE E4 A1          test al,dh      ;look at bit for this r
26F0 84 C6          ret              ;return status of ready
26F2 C3

;*****
;*
;* Send_com sends a command and parameters *
;* to the 18271: BX addresses parameters. *
;*****

```

All Information Presented Here is Proprietary to Digital Research

```

; * The DMA controller is also initialized *
; * if this is a read or write *
; *
; *****
send_com:
26F3 E4 A0      in al,fdc_stat
26F5 A8 80      test al,80h      ;insure command not bus
26F7 75 FA      jnz send_com      ;loop until ready

                ;see if we have to initialize for a DMA

26F9 8A 47 01   mov al,1[bx]      ;get command byte
26FC 3C 12      cmp al,12h
26FE 75 04      jne write_maybe ;if not a read it could
2700 B1 40      mov cl,40h
2702 EB 06      jmps init_dma     ;is a read command, go
write_maybe:
2704 3C 0A      cmp al,0ah
2706 75 20      jne dma_exit      ;leave DMA alone if not
2708 B1 80      mov cl,80h        ;we have write, not rea

init_dma:
;we have a read or write operation, setup DMA c
; (CL contains proper direction bit)
270A 90 04      mov al,04h
270C E6 A8      out dmact_mode,al ;enable dmact
270E B0 00      mov al,00
2710 E6 A5      out dmact_cont,al ;send first byte to
2712 8A C1      mov al,cl
2714 E6 A5      out dmact_cont,al ;load direction regi
2716 A1 65 28   mov ax,dma_adr
2719 E6 A4      out dmact_adr,al ;send low byte of DM
271B 8A C4      mov al,ah
271D E6 A4      out dmact_adr,al ;send high byte
271F A1 67 28   mov ax,dma_seg
2722 E6 AA      out fdc_segment,al ;send low byte of se
2724 8A C4      mov al,ah
2726 E6 AA      out fdc_segment,al ;then high segment a

dma_exit:
2728 8A 0F      mov cl,[BX]      ;get count
272A 43         inc BX
272B 8A 07      mov al,[BX]      ;get command
272D 0A 06 69 28 or al,sel_mask ;merge command and driv
2731 E6 A0      out fdc_com,al ;send command byte

parm_loop:
2733 FE C9      dec cl
2735 74 82      jz exec_exit      ;no (more) parameters,
2737 43         inc BX           ;point to (next) parame

parm_poll:
2738 E4 A0      in al,fdc_stat
273A A8 20      test al,20h      ;test "parameter regist
273C 75 FA      jnz parm_poll    ;idle until parm reg no
273E 8A 07      mov al,[BX]
2740 E6 A1      out fdc_parm,al ;send next parameter
2742 EB EF      jmps parm_loop ;go see if there are mo

```

```

;*****
;*
;*          Data Areas
;*
;*****
2744      data_offset      equ offset $

          dseg
          org      data_offset      ;contiguous with
          IF      loader_bios
;-----
;|
signon    db      cr,lf,cr,lf
          db      'CP/M-86 Version 2.2',cr,lf,0
;|
;-----
          ENDIF      ;loader_bios

          IF      not loader_bios
;-----
;|
2744      0D 0A 0D 0A      signon    db      cr,lf,cr,lf
2748      20 20 53 79 73 74      db      'System Generated - 11 Jan 8
          65 6D 20 47 65 6E
          65 72 61 74 65 64
          20 20 2D 20 31 31
          20 4A 61 6E 20 38
          31 0D 0A 00
;|
;-----
          ENDIF      ;not loader_bios

276A      0D 0A 48 6F 6D 65      bad_hom db      cr,lf,'Home Error',cr,lf,0
          20 45 72 72 6F 72
          0D 0A 00
2779      0D 0A 49 6E 74 65      int_trp db      cr,lf,'Interrupt Trap Halt',cr,
          72 72 75 70 74 20
          54 72 61 70 20 48
          61 6C 74 0D 0A 00

2791      B1 27 B1 27 B1 27      errtbl dw er0,er1,er2,er3
          B1 27
2799      C1 27 D1 27 DE 27      dw er4,er5,er6,er7
          EF 27
27A1      02 28 16 28 28 28      dw er8,er9,erA,erB
          3D 28
27A9      4D 28 B1 27 B1 27      dw erC,erD,erE,erF
          B1 27

27B1      0D 0A 4E 75 6C 6C      er0      db      cr,lf,'Null Error ??',0
          20 45 72 72 6F 72
          20 3F 3F 00
27B1      er1      equ er0

```

All Information Presented Here is Proprietary to Digital Research

```

27B1      er2      equ er0
27B1      er3      equ er0
27C1 0D 0A 43 6C 6F 63 er4      db  cr,lf,'Clock Error :',0
      6B 20 45 72 72 6F
      72 20 3A 00
27D1 0D 0A 4C 61 74 65 er5      db  cr,lf,'Late DMA :',0
      20 44 4D 41 20 3A
      00
27DE 0D 0A 49 44 20 43 er6      db  cr,lf,'ID CRC Error :',0
      52 43 20 45 72 72
      6F 72 20 3A 00
27EF 0D 0A 44 61 74 61 er7      db  cr,lf,'Data CRC Error :',0
      20 43 52 43 20 45
      72 72 6F 72 20 3A
      00
2802 0D 0A 44 72 69 76 er8      db  cr,lf,'Drive Not Ready :',0
      65 20 4E 6F 74 20
      52 65 61 64 79 20
      3A 00
2816 0D 0A 57 72 69 74 er9      db  cr,lf,'Write Protect :',0
      65 20 50 72 6F 74
      65 63 74 20 3A 00
2828 0D 0A 54 72 6B 20 erA      db  cr,lf,'Trk 00 Not Found :',0
      30 30 20 4E 6F 74
      20 46 6F 75 6E 64
      20 3A 00
283D 0D 0A 57 72 69 74 erB      db  cr,lf,'Write Fault :',0
      65 20 46 61 75 6C
      74 20 3A 00
284D 0D 0A 53 65 63 74 erC      db  cr,lf,'Sector Not Found :',0
      6F 72 20 4E 6F 74
      20 46 6F 75 6E 64
      20 3A 00

27B1      erD      equ er0
27B1      erE      equ er0
27B1      erF      equ er0
2802      nrdymsg equ er8

2862 00      rtry_cnt db 0      ;disk error retry counter
2863 00 00    last_com dw 0      ;address of last command string
2865 00 00    dma_adr  dw 0      ;dma offset stored here
2867 00 00    dma_seg  dw 0      ;dma segment stored here
2869 40      sel_mask db 40h    ;select mask, 40h or 80h

;          Various command strings for i8271

286A 03      io_com  db 3      ;length
286B 00      rd_wr   db 0      ;read/write function code
286C 00      trk     db 0      ;track #
286D 00      sect    db 0      ;sector #

286E 02 29 00 hom_com db 2,29h,0 ;home drive command
2871 01 2C    rds_com db 1,2ch   ;read status command

;          System Memory Segment Table

```

All Information Presented Here is Proprietary to Digital Research

```

2873 02          segtable db 2      ;2 segments
2874 DF 02          dw tpa_seg      ;1st seg starts after B
2876 21 05          dw tpa_len      ;and extends to 08000
2878 00 20          dw 2000h        ;second is 20000 -
287A 00 20          dw 2000h        ;3FFFF (128k)

=
=          include singles.lib ;read in disk defin
=          ; DISKS 2
=          287C      dpbase equ      S          ;Base of Disk P
=          287C AB 28 00 00      dpe0 dw      xlt0,0000h      ;Translate Tabl
=          2880 00 00 00 00      dw      0000h,0000h        ;Scratch Area
=          2884 C5 28 9C 28      dw      dirbuf,dpb0         ;Dir Buff, Parm
=          2888 64 29 45 29      dw      csv0,alv0           ;Check, Alloc V
=          288C AB 28 00 00      dpe1 dw      xlt1,0000h      ;Translate Tabl
=          2890 00 00 00 00      dw      0000h,0000h        ;Scratch Area
=          2894 C5 28 9C 28      dw      dirbuf,dpb1         ;Dir Buff, Parm
=          2898 93 29 74 29      dw      csv1,alv1           ;Check, Alloc V
=          ; DISKDEF 0,1,26,6,1024,243,64,64
=          289C      dpb0 equ      offset S      ;Disk Parameter
=          289C 1A 00      dw      26              ;Sectors Per Tr
=          289E 03      db      3              ;Block Shift
=          289F 07      db      7              ;Block Mask
=          28A0 00      db      0              ;Extnt Mask
=          28A1 F2 00      dw      242           ;Disk Size - 1
=          28A3 3F 00      dw      63           ;Directory Max
=          28A5 C0      db      192            ;Alloc0
=          28A6 00      db      0              ;Alloc1
=          28A7 10 00      dw      16           ;Check Size
=          28A9 02 00      dw      2            ;Offset
=          28AB      xlt0 equ      offset S      ;Translate Tabl
=          28AB 01 07 0D 13      db      1,7,13,19
=          28AF 19 05 0B 11      db      25,5,11,17
=          28B3 17 03 09 0F      db      23,3,9,15
=          28B7 15 02 08 0E      db      21,2,8,14
=          28BB 14 1A 06 0C      db      20,26,6,12
=          28BF 12 18 04 0A      db      18,24,4,10
=          28C3 10 16      db      16,22
=          001F      als0 equ      31              ;Allocation Vec
=          0010      css0 equ      16              ;Check Vector S
=          ; DISKDEF 1,0
=          289C      dpb1 equ      dpb0          ;Equivalent Par
=          001F      als1 equ      als0          ;Same Allocatio
=          0010      css1 equ      css0          ;Same Checksum
=          28AB      xlt1 equ      xlt0          ;Same Translate
=          ; ENDEF
=          ;
=          ; Uninitialized Scratch Memory Follows:
=          ;
=          28C5      begdat equ      offset S      ;Start of Scrat
=          28C5      dirbuf rs      128          ;Directory Buff
=          2945      alv0 rs      als0          ;Alloc Vector
=          2964      csv0 rs      css0          ;Check Vector
=          2974      alv1 rs      als1          ;Alloc Vector
=          2993      csv1 rs      css1          ;Check Vector

```

All Information Presented Here is Proprietary to Digital Research

```

= 29A3      enddat equ offset $      ;End of Scratch
= 00DE      datsiz equ offset $-begdat ;Size of Scratch
= 29A3 00    db 0                    ;Marks End of M

29A4        loc_stk rw 32 ;local stack for initialization
29E4        stkbases equ offset $

29E4        lastoff equ offset $
02DF        tpa_seg equ (lastoff+0400h+15) / 16
0521        tpa_len equ 0800h - tpa_seg
29E4 00     db 0 ;fill last address for GENCMD

;*****
;*
;*          Dummy Data Section
;*
;*****
0000        dseg 0 ;absolute low memory
            org 0 ;(interrupt vectors)
0000        int0_offset rw 1
0002        int0_segment rw 1
;          pad to system call vector
0004        rw 2*(bdos_int-1)

0380        bdos_offset rw 1
0382        bdos_segment rw 1
            END

```

## APPENDIX E: CBIOS LISTING

```
*****
*
* This is the listing of the skeletal CBIOS which
* you can use as the basis for a customized BIOS
* for non-standard hardware. The essential por-
* tions of the BIOS remain, with "rs" statements
* marking the routines to be inserted.
*
*****
```

```
*****
;
; * This Customized BIOS adapts CP/M-86 to
; * the following hardware configuration
; *   Processor:
; *   Brand:
; *   Controller:
; *
; *   Programmer:
; *   Revisions :
; *
*****
```

```
FFFF      true          equ -1
0000      false        equ not true
000D      cr           equ 0dh ;carriage return
000A      lf           equ 0ah ;line feed
```

```
*****
;
; * Loader_bios is true if assembling the
; * LOADER_BIOS, otherwise BIOS is for the
; * CPM.SYS file.
; *
*****
```

```
0000      loader_bios   equ false
00E0      bdos_int      equ 224 ;reserved BDOS interrup
```

```
          IF          not loader_bios
```

```
2500      bios_code     equ 2500h
0000      ccp_offset    equ 0000h
0B06      bdos_ofst     equ 0B06h ;BDOS entry point
```

```
          ENDIF      ;not loader_bios
```

All Information Presented Here is Proprietary to Digital Research

```

                                IF      loader_bios
;-----
;|
bios_code      equ 1200h ;start of LDBIOS
ccp_offset     equ 0003h ;base of CPMLOADER
bdos_ofst      equ 0406h ;stripped BDOS entry
;|
;-----
                                ENDIF      ;loader_bios

                                cseq
                                org      ccpoffset
ccp:
                                org      bios_code

;*****
;*
;* BIOS Jump Vector for Individual Routines *
;*
;*****

2500 E9 3C 00      jmp INIT      ;Enter from BOOT ROM or LOADER
2503 E9 79 00      jmp WBOOT     ;Arrive here from BDOS call 0
2506 E9 85 00      jmp CONST     ;return console keyboard status
2509 E9 8D 00      jmp CONIN     ;return console keyboard char
250C E9 9A 00      jmp CONOUT    ;write char to console device
250F E9 A2 00      jmp LISTOUT   ;write character to list device
2512 E9 B5 00      jmp PUNCH     ;write character to punch device
2515 E9 BD 00      jmp READER     ;return char from reader device
2518 E9 F6 00      jmp HOME      ;move to trk 00 on cur sel driv
251B E9 D9 00      jmp SELDSK    ;select disk for next rd/write
251E E9 01 01      jmp SETTRK    ;set track for next rd/write
2521 E9 03 01      jmp SETSEC    ;set sector for next rd/write
2524 E9 0C 01      jmp SETDMA    ;set offset for user buff (DMA)
2527 E9 17 01      jmp READ      ;read a 128 byte sector
252A E9 47 01      jmp WRITE     ;write a 128 byte sector
252D E9 8F 00      jmp LISTST    ;return list status
2530 E9 F9 00      jmp SECTTRAN  ;xlate logical->physical sector
2533 E9 02 01      jmp SETDMAB   ;set seg base for buff (DMA)
2536 E9 04 01      jmp GETSEGT   ;return offset of Mem Desc Tabl
2539 E9 A4 00      jmp GETIOBF   ;return I/O map byte (IOBYTE)
253C E9 A5 00      jmp SETIOBF   ;set I/O map byte (IOBYTE)

;*****
;*
;* INIT Entry Point, Differs for LDBIOS and *
;* BIOS, according to "Loader_Bios" value *
;*
;*****

INIT:      ;print signon message and initialize ha
253F 8C C8      mov ax,cs      ;we entered with a JMPF
2541 8E D0      mov ss,ax      ;CS: as the initial val
2543 8E D8      mov ds,ax      ;DS:,

```

## CBIOS Listing

## Appendix E

```

2545 8E C0      mov es,ax      ;and ES:
                ;use local stack during initialization
2547 BC 59 28   mov sp,offset stkbases
254A FC        cld            ;set forward direction

                IF      not loader_bios
;-----
;|
;|      ; This is a BIOS for the CPM.SYS file.
;|      ; Setup all interrupt vectors in low
;|      ; memory to address trap
;|
254B 1E        push ds        ;save the DS register
254C C6 06 A7 26 00   mov IOBYTE,0      ;clear IOBYTE
2551 B8 00 00       mov ax,0
2554 8E D8        mov ds,ax
2556 8E C0        mov es,ax      ;set ES and DS to zero
                ;setup interrupt 0 to address trap rout
2558 C7 06 00 00 82 25   mov int0_offset,offset int_trap
255E 8C 0E 02 00     mov int0_segment,CS
2562 BF 04 00       mov di,4
2565 BE 00 00       mov si,0      ;then propagate
2568 B9 FE 01       mov cx,510    ;trap vector to
256B F3 A5         rep movs ax,ax ;all 256 interrupts
                ;BDOS offset to proper interrupt
                mov bdos_offset,bdos_ofst
256D C7 06 80 03 06 0B   mov bdos_offset,bdos_ofst
2573 1F          pop ds        ;restore the DS register

;      (additional CP/M-86 initialization)
;|
;-----
                ENDIF      ;not loader_bios

                IF      loader_bios
;-----
;|
;|      ;This is a BIOS for the LOADER
;|      push ds          ;save data segment
;|      mov ax,0
;|      mov ds,ax        ;point to segment zero
;|      ;BDOS interrupt offset
;|      mov bdos_offset,bdos_ofst
;|      mov bdos_segment,CS ;bdos interrupt seg
;|      (additional LOADER initialization)
;|      pop ds          ;restore data segment
;|
;-----
                ENDIF      ;loader_bios

2574 BB B1 26      mov bx,offset signon
2577 E8 6F 00      call pmsg      ;print signon message
257A B1 00         mov cl,0       ;default to dr A: on co
257C E9 81 DA      jmp ccp        ;jump to cold start ent

257F E9 84 DA      WBOOT: jmp ccp+6      ;direct entry to CCP at

```

All Information Presented Here is Proprietary to Digital Research

```

                                IF      not loader_bios
                                -----
                                ;!
int_trap:
2582 FA      cli                ;block interrupts
2583 8C C8    mov ax,cs
2585 8E D8    mov ds,ax        ;get our data segment
2587 BB D1 26  mov bx,offset int_trp
258A E8 5C 00  call pmsg
258D F4      hlt                ;hardstop
                                ;!
                                -----
                                ENDIF    ;not loader_bios

                                ;*****
                                ;*
                                ;*  CP/M Character I/O Interface Routines  *
                                ;*
                                ;*****

CONST:
258E      rs                ;console status
2598 C3     ret             10      ;((fill-in))

CONIN:
2599 E8 F2 FF call CONST
259C 74 FB    jz CONIN        ;wait for RDA
259E      rs                ;((fill-in))
25A8 C3     ret             10

CONOUT:
25A9      rs                ;console output
25B3 C3     ret             10      ;((fill-in))
                                ;then return data

LISTOUT:
25B4      rs                ;list device output
25BE C3     ret             10      ;((fill-in))

LISTST:
25BF      rs                ;poll list status
25C9 C3     ret             10      ;((fill-in))

PUNCH:
25CA      rs                ;write punch device
25D4 C3     ret             10      ;((fill-in))

READER:
25D5      rs                ;((fill-in))
25DF C3     ret             10

GETIOBF:
25E0 A0 A7 26 mov al,IOBYTE
25E3 C3     ret

```

All Information Presented Here is Proprietary to Digital Research

```

25E4 88 0E A7 26      SETIOBF:      mov IOBYTE,cl      ;set iobyte
25E8 C3                ret                ;iobyte not implemented

pmsg:
25E9 8A 07            mov al,[BX]          ;get next char from mes
25EB 84 C0            test al,al
25ED 74 21            jz return            ;if zero return
25EF 8A C8            mov CL,AL
25F1 E8 B5 FF        call CONOUT          ;print it
25F4 43                inc BX
25F5 EB F2            jmps pmsg           ;next character and loop

;*****
;*
;*          Disk Input/Output Routines
;*
;*****

0002
25F7 88 0E A8 26      SELDSK:      ;select disk given by register
ndisks equ 2 ;number of disks (up to 16)
mov disk,cl          ;save disk number
25FB BB 00 00        mov bx,0000h        ;ready for error return
25FE 80 F9 02        cmp cl,ndisks      ;n beyond max disks?
2601 73 0D            jnb return        ;return if so
2603 B5 00            mov ch,0          ;double(n)
2605 8B D9            mov bx,cx         ;bx = n
2607 B1 04            mov cl,4          ;ready for *16
2609 D3 E3            shl bx,cl         ;n = n * 16
260B B9 F1 26        mov cx,offset dpbase
260E 03 D9            add bx,cx         ;dpbase + n * 16
2610 C3                return: ret      ;bx = .doh

HOME: ;move selected disk to home position (T
2611 C7 06 A9 26 00 00 mov trk,0      ;set disk i/o to track
2617                rs 10             ;((fill-in)
2621 C3                ret

2622 89 0E A9 26      SETTRK: ;set track address given by CX
2626 C3                mov trk,CX
ret

2627 89 0E AB 26      SETSEC: ;set sector number given by cx
262B C3                mov sect,CX
ret

262C 8B D9            SECTTRAN: ;translate sector CX using table at [D
262E 03 DA            mov bx,cx
2630 8A 1F            add bx,dx          ;add sector to tran tab
2632 C3                mov bl,[bx]       ;get logical sector
ret

2633 89 0E AD 26      SETDMA: ;set DMA offset given by CX
2637 C3                mov dma_adr,CX
ret

```

All Information Presented Here is Proprietary to Digital Research

```

;-----
ENDIF ;loader_bios

IF not loader_bios
;-----
;|
26B1 0D 0A 0D 0A signon db cr,lf,cr,lf
26B5 53 79 73 74 65 6D db 'System Generated 00/00/00'
      20 47 65 6E 65 72
      61 74 65 64 20 30
      30 2F 30 30 2F 30
      30
26CE 0D 0A 00 db cr,lf,0
;|
;-----
ENDIF ;not loader_bios

26D1 0D 0A int_trp db cr,lf
26D3 49 6E 74 65 72 72 db 'Interrupt Trap Halt'
      75 70 74 20 54 72
      61 70 20 48 61 6C
      74
26E6 0D 0A db cr,lf

; System Memory Segment Table

26E8 02 segtable db 2 ;2 segments
26E9 C6 02 dw tpa_seg ;1st seg starts after B
26EB 3A 05 dw tpa_len ;and extends to 08000
26ED 00 20 dw 2000h ;second is 20000 -
26EF 00 20 dw 2000h ;3FFFF (128k)

= include singles.lib ;read in disk defin
= ; DISKS 2
= 26F1 dpbase equ 5 ;Base of Disk P
= 26F1 20 27 00 00 dpe0 dw xlt0,0000h ;Translate Tabl
= 26F5 00 00 00 00 dw 0000h,0000h ;Scratch Area
= 26F9 3A 27 11 27 dw dirbuf,dpb0 ;Dir Buff, Parm
= 26FD D9 27 BA 27 dw csv0,alv0 ;Check, Alloc V
= 2701 20 27 00 00 dpe1 dw xlt1,0000h ;Translate Tabl
= 2705 00 00 00 00 dw 0000h,0000h ;Scratch Area
= 2709 3A 27 11 27 dw dirbuf,dpb1 ;Dir Buff, Parm
= 270D 08 28 E9 27 dw csv1,alv1 ;Check, Alloc V
= ; DISKDEF 0,1,26,6,1024,243,64,64
= 2711 dpb0 equ offset S ;Disk Parameter
= 2711 1A 00 dw 26 ;Sectors Per Tr
= 2713 03 db 3 ;Block Shift
= 2714 07 db 7 ;Block Mask
= 2715 00 db 0 ;Extnt Mask
= 2716 F2 00 dw 242 ;Disk Size - 1
= 2718 3F 00 dw 63 ;Directory Max
= 271A C0 db 192 ;Alloc0
= 271B 00 db 0 ;Alloc1
= 271C 10 00 dw 16 ;Check Size
= 271E 02 00 dw 2 ;Offset

```

All Information Presented Here is Proprietary to Digital Research

## CBIOS Listing

## Appendix E

```

= 2720          xlt0      equ      offset $          ;Translate Tabl
= 2720 01 07 0D 13      db      1,7,13,19
= 2724 19 05 0B 11      db      25,5,11,17
= 2728 17 03 09 0F      db      23,3,9,15
= 272C 15 02 08 0E      db      21,2,8,14
= 2730 14 1A 06 0C      db      20,26,6,12
= 2734 12 18 04 0A      db      18,24,4,10
= 2738 10 16           db      16,22
= 001F          als0      equ      31                ;Allocation Vec
= 0010          css0      equ      16                ;Check Vector S
=                ; DISKDEF 1,0
= 2711          dpb1      equ      dpb0              ;Equivalent Par
= 001F          als1      equ      als0              ;Same Allocatio
= 0010          css1      equ      css0              ;Same Checksum
= 2720          xlt1      equ      xlt0              ;Same Translate
=                ; ENDEF
=                ;
=                ; Uninitialized Scratch Memory Follows:
=                ;
= 273A          begdat     equ      offset $          ;Start of Scrat
= 273A          dirbuf     rs      128              ;Directory Buff
= 27BA          alv0       rs      als0              ;Alloc Vector
= 27D9          csv0       rs      css0              ;Check Vector
= 27E9          alv1       rs      als1              ;Alloc Vector
= 2808          csv1       rs      css1              ;Check Vector
= 2818          enddat     equ      offset $          ;End of Scratch
= 00DE          datsiz     equ      offset $-begdat   ;Size of Scratch
= 2818 00         db      0                        ;Marks End of M

2819          loc_stk     rw      32                ;local stack for initialization
2859          stkBase     equ      offset $

2859          lastoff     equ      offset $
02C6          tpa_seg     equ      (lastoff+0400h+15) / 16
053A          tpa_len     equ      0800h - tpa_seg
2859 00         db      0                        ;fill last address for GENCMD

;*****
;*
;*          Dummy Data Section
;*
;*****
0000          dseg        0                ;absolute low memory
              org         0                ;(interrupt vectors)
0000          int0_offset  rw         1
0002          int0_segment rw         1
;          pad to system call vector
0004          rw          2*(bdos_int-1)

0380          bdos_offset  rw         1
0382          bdos_segment rw         1
              END

```

All Information Presented Here is Proprietary to Digital Research

# APPENDIX F: BLOCKING AND DEBLOCKING ALGORITHMS

```

1: ;*****
2: ;*
3: ;*      Sector Blocking / Deblocking
4: ;*
5: ;* This algorithm is a direct translation of the
6: ;* CP/M-80 Version, and is included here for refer-
7: ;* ence purposes only. The file DEBLOCK.LIB is in-
8: ;* cluded on your CP/M-86 disk, and should be used
9: ;* for actual applications. You may wish to contact
10: ;* Digital Research for notices of updates.
11: ;*
12: ;*****
13: ;
14: ;*****
15: ;*
16: ;*      CP/M to host disk constants
17: ;*
18: ;* (This example is setup for CP/M block size of 16K
19: ;* with a host sector size of 512 bytes, and 12 sec-
20: ;* tors per track. Blksiz, hstsiz, hstspt, hstblk
21: ;* and secshf may change for different hardware.)
22: ;*****
23: una      equ      byte ptr [BX]      ;name for byte at BX
24: ;
25: blksiz   equ      16384              ;CP/M allocation size
26: hstsiz   equ      512                ;host disk sector size
27: hstspt   equ      12                ;host disk sectors/trk
28: hstblk   equ      hstsiz/128        ;CP/M sects/host buff
29: ;
30: ;*****
31: ;*
32: ;* secshf is log2(hstblk), and is listed below for
33: ;* values of hstsiz up to 2048.
34: ;*
35: ;*      hstsiz      hstblk      secshf
36: ;*      256         2          1
37: ;*      512         4          2
38: ;*      1024        8          3
39: ;*      2048        16         4
40: ;*
41: ;*****
42: secshf   equ      2                  ;log2(hstblk)
43: cpmspt   equ      hstblk * hstspt    ;CP/M sectors/track
44: secmsk   equ      hstblk-1          ;sector mask
45: ;
46: ;*****
47: ;*
48: ;*      BDOS constants on entry to write
49: ;*
50: ;*****

```

All Information Presented Here is Proprietary to Digital Research

```

51: wrall    equ    0           ;write to allocated
52: wrdir    equ    1           ;write to directory
53: wrual    equ    2           ;write to unallocated
54: ;
55: ;*****
56: ;*
57: ;*      The BIOS entry points given below show the      *
58: ;*      code which is relevant to deblocking only.      *
59: ;*
60: ;*****
61: seldsk:
62:           ;select disk
63:           ;is this the first activation of the drive?
64:           test DL,1           ;lsb = 0?
65:           jnz selset
66:           ;this is the first activation, clear host buff
67:           mov hstact,0
68:           mov unacnt,0
69: selset:
70:           mov al,cl ! cbw      ;put in AX
71:           mov sekdisk,al      ;seek disk number
72:           mov cl,4 ! shl al,cl ;times 16
73:           add ax,offset dpbase
74:           mov bx,ax
75:           ret
76: ;
77: home:
78:           ;home the selected disk
79:           mov al,hstwrtr      ;check for pending write
80:           test al,al
81:           jnz homed
82:           mov hstact,0        ;clear host active flag
83: homed:
84:           mov cx,0            ;now, set track zero
85:           ; (continue HOME routine)
86:           ret
87: ;
88: settrk:
89:           ;set track given by registers CX
90:           mov sektrk,CX       ;track to seek
91:           ret
92: ;
93: setsec:
94:           ;set sector given by register cl
95:           mov seksec,cl       ;sector to seek
96:           ret
97: ;
98: setdma:
99:           ;set dma address given by CX
100:          mov dma_off,CX
101:          ret
102: ;
103: setdmab:
104:          ;set segment address given by CX
105:          mov dma_seq,CX

```

All Information Presented Here is Proprietary to Digital Research

```

106:         ret
107: ;
108: sectran:
109:         ;translate sector number CX with table at [DX]
110:         test DX,DX           ;test for hard skewed
111:         jz notran           ;(blocked must be hard skewed)
112:         mov BX,CX
113:         add BX,DX
114:         mov BL,[BX]
115:         ret
116: no_tran:
117:         ;hard skewed disk, physical = logical sector
118:         mov BX,CX
119:         ret
120: ;
121: read:
122:         ;read the selected CP/M sector
123:         mov unacnt,0         ;clear unallocated counter
124:         mov readop,1         ;read operation
125:         mov rsflag,1         ;must read data
126:         mov wrtype,wrual     ;treat as unalloc
127:         jmp rwoper           ;to perform the read
128: ;
129: write:
130:         ;write the selected CP/M sector
131:         mov readop,0         ;write operation
132:         mov wrtype,cl
133:         cmp cl,wrual         ;write unallocated?
134:         jnz chkuna          ;check for unalloc
135: ;
136: ;       write to unallocated, set parameters
137: ;
138:         mov unacnt,(blksiz/128) ;next unalloc recs
139:         mov al,sekdisk        ;disk to seek
140:         mov unadsk,al         ;unadsk = sekdisk
141:         mov ax,sektrk
142:         mov unatr,ax          ;unatr = sektrk
143:         mov al,seksec
144:         mov unasec,al         ;unasec = seksec
145: ;
146: chkuna:
147:         ;check for write to unallocated sector
148: ;
149:         mov bx,offset unacnt   ;point "UNA" at UNACNT
150:         mov al,una ! test al,al ;any unalloc remain?
151:         jz alloc              ;skip if not
152: ;
153: ;       more unallocated records remain
154:         dec al                 ;unacnt = unacnt-1
155:         mov una,al
156:         mov al,sekdisk        ;same disk?
157:         mov BX,offset unadsk
158:         cmp al,una            ;sekdisk = unadsk?
159:         jnz alloc             ;skip if not
160: ;

```

All Information Presented Here is Proprietary to Digital Research

```

161: ;      disks are the same
162:      mov AX, unatrck
163:      cmp AX, sektrk
164:      jnz alloc          ;skip if not
165: ;
166:      tracks are the same
167:      mov al,seksec      ;same sector?
168: ;
169:      mov BX,offset unasec ;point una at unasec
170: ;
171:      cmp al,una         ;seksec = unasec?
172:      jnz alloc          ;skip if not
173: ;
174:      match, move to next sector for future ref
175:      inc una            ;unasec = unasec+1
176:      mov al,una         ;end of track?
177:      cmp al,cpmspt      ;count CP/M sectors
178:      jb noovf           ;skip if below
179: ;
180:      overflow to next track
181:      mov una,0          ;unasec = 0
182:      inc unatrck        ;unatrck=unatrck+1
183: ;
184: noovf:
185:      ;match found, mark as unnecessary read
186:      mov rsflag,0       ;rsflag = 0
187:      jmps rwoper        ;to perform the write
188: ;
189: alloc:
190:      ;not an unallocated record, requires pre-read
191:      mov unacnt,0       ;unacnt = 0
192:      mov rsflag,1       ;rsflag = 1
193:      ;drop through to rwoper
194: ;
195: ;*****
196: ;*
197: ;*      Common code for READ and WRITE follows
198: ;*
199: ;*****
200: rwoper:
201:      ;enter here to perform the read/write
202:      mov erflag,0       ;no errors (yet)
203:      mov al, seksec     ;compute host sector
204:      mov cl, secshf
205:      shr al,cl
206:      mov sekfst,al      ;host sector to seek
207: ;
208: ;      active host sector?
209:      mov al,1
210:      xchg al,hstact      ;always becomes 1
211:      test al,al         ;was it already?
212:      jz filhst          ;fill host if not
213: ;
214: ;      host buffer active, same as seek buffer?
215:      mov al,sekdst

```

All Information Presented Here is Proprietary to Digital Research

```

216:      cmp al,hstdsk      ;sekdisk = hstdsk?
217:      jnz nomatch
218: ;
219: ;      same disk, same track?
220:      mov ax,hsttrk
221:      cmp ax,sektrk      ;host track same as seek track
222:      jnz nomatch
223: ;
224: ;      same disk, same track, same buffer?
225:      mov al,sekhst
226:      cmp al,hstsec      ;sekhst = hstsec?
227:      jz match           ;skip if match
228: nomatch:
229:      ;proper disk, but not correct sector
230:      mov al,hstwrtr
231:      test al,al         ;"dirty" buffer ?
232:      jz filhst         ;no, don't need to write
233:      call writehst      ;yes, clear host buff
234: ;      (check errors here)
235: ;
236: filhst:
237:      ;may have to fill the host buffer
238:      mov al,sekdisk ! mov hstdsk,al
239:      mov ax,sektrk ! mov hsttrk,ax
240:      mov al,sekhst ! mov hstsec,al
241:      mov al,rsflag
242:      test al,al         ;need to read?
243:      jz filhstl
244: ;
245:      call readhst      ;yes, if 1
246: ;      (check errors here)
247: ;
248: filhstl:
249:      mov hstwrtr,0      ;no pending write
250: ;
251: match:
252:      ;copy data to or from buffer depending on "readop"
253:      mov al,seksec      ;mask buffer number
254:      and ax,secmsk      ;least signif bits are masked
255:      mov cl,7 ! shl ax,cl ;shift left 7 (* 128 = 2**7)
256: ;
257: ;      ax has relative host buffer offset
258: ;
259:      add ax,offset hstbuf ;ax has buffer address
260:      mov si,ax          ;put in source index register
261:      mov di,dma_off     ;user buffer is dest if readop
262: ;
263:      push DS ! push ES  ;save segment registers
264: ;
265:      mov ES,dma_seg     ;set destseg to the users seg
266:      ;SI/DI and DS/ES is swapped
267:      ;if write op
268:      mov cx,128/2       ;length of move in words
269:      mov al,readop
270:      test al,al         ;which way?

```

All Information Presented Here is Proprietary to Digital Research

```

271:      jnz      rwmov     ;skip if read
272: ;
273: ;      write operation, mark and switch direction
274:      mov hstwr,1      ;hstwr = 1 (dirty buffer now)
275:      xchg si,di      ;source/dest index swap
276:      mov ax,DS
277:      mov ES,ax
278:      mov DS,dma_seg      ;setup DS,ES for write
279: ;
280: rwmov:
281:      cld ! rep movs AX,AX      ;move as 16 bit words
282:      pop ES ! pop DS      ;restore segment registers
283: ;
284: ;      data has been moved to/from host buffer
285:      cmp wrtype,wrdir      ;write type to directory?
286:      mov al,erflag      ;in case of errors
287:      jnz return_rw      ;no further processing
288: ;
289: ;      clear host buffer for directory write
290:      test al,al      ;errors?
291:      jnz return_rw      ;skip if so
292:      mov hstwr,0      ;buffer written
293:      call writehst
294:      mov al,erflag
295: return_rw:
296:      ret
297: ;
298: ;*****
299: ;*
300: ;* WRITEHST performs the physical write to the host
301: ;* disk, while READHST reads the physical disk.
302: ;*
303: ;*****
304: writehst:
305:      ret
306: ;
307: readhst:
308:      ret
309: ;
310: ;*****
311: ;*
312: ;* Use the GENDEF utility to create disk def tables
313: ;*
314: ;*****
315: dibase equ offset s
316: ;      disk parameter tables go here
317: ;
318: ;*****
319: ;*
320: ;* Uninitialized RAM areas follow, including the
321: ;* areas created by the GENDEF utility listed above.
322: ;*
323: ;*****
324: sek_dsk rb      1      ;seek disk number
325: sek_trk rw      1      ;seek track number

```

All Information Presented Here is Proprietary to Digital Research

```
326: sek_sec  rb      1      ;seek sector number
327: ;
328: hst_dsk  rb      1      ;host disk number
329: hst_trk  rw      1      ;host track number
330: hst_sec  rb      1      ;host sector number
331: ;
332: sek_hst  rb      1      ;seek shr secshf
333: hst_act  rb      1      ;host active flag
334: hst_wrt  rb      1      ;host written flag
335: ;
336: una_cnt  rb      1      ;unalloc rec cnt
337: una_dsk  rb      1      ;last unalloc disk
338: una_trk  rw      1      ;last unalloc track
339: una_sec  rb      1      ;last unalloc sector
340: ;
341: erflag   rb      1      ;error reporting
342: rsflag   rb      1      ;read sector flag
343: readop   rb      1      ;1 if read operation
344: wrtype   rb      1      ;write operation type
345: dma_seg  rw      1      ;last dma segment
346: dma_off  rw      1      ;last dma offset
347: hstBuf   rb      hstsiz ;host buffer
348:          end
```